Data Science

Mining Data Streams

Themis Palpanas
University of Paris

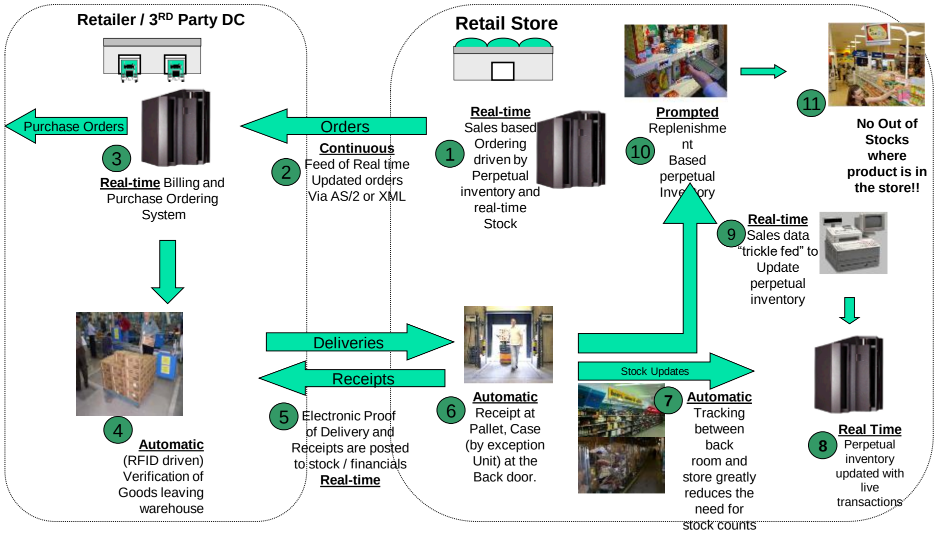
1

Thanks for slides to:

- Minos Garofalakis
- Divesh Srivastava
- Nick Koudas
- Jiawei Han
- Jeffrey Ullman
- Anand Rajaraman

2

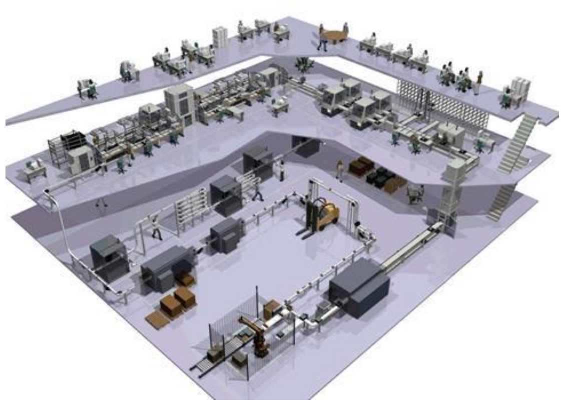
Motivating Examples: Store Replenishment Process



Data Science 3

3

Motivating Examples: Production Control System

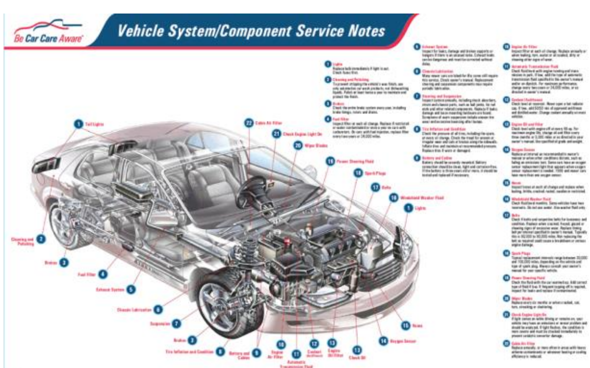


Data Science 6

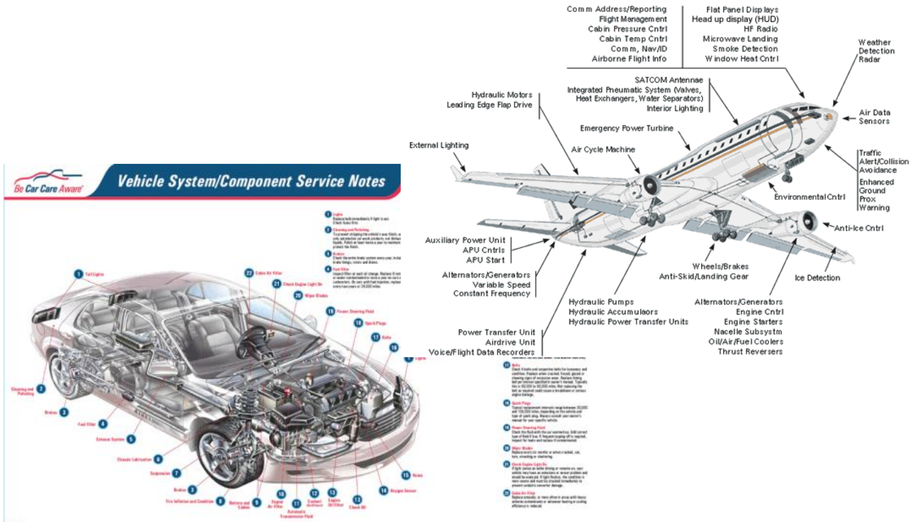
6

Motivating Examples: Production Control System

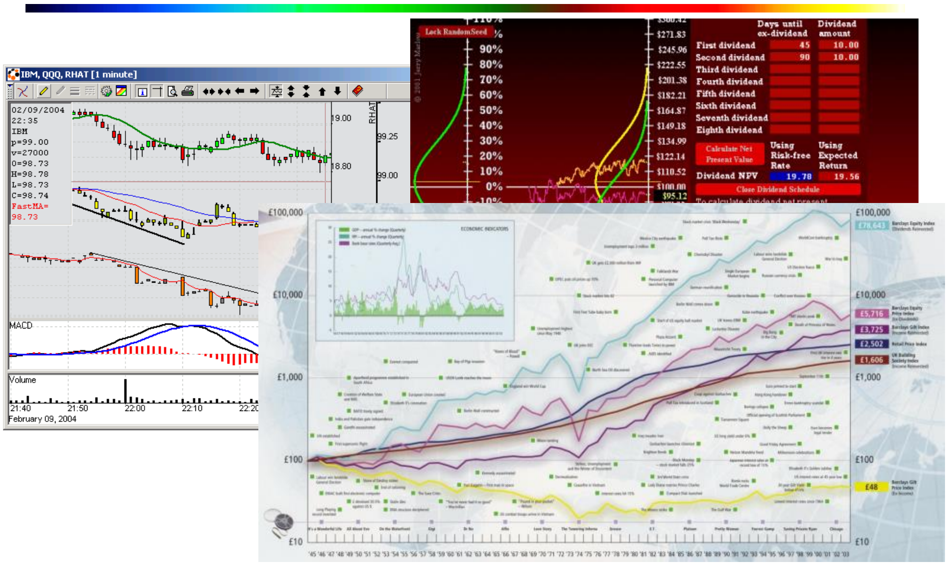
Motivating Examples: Monitoring Vehicle Operation



Motivating Examples: Monitoring Vehicle Operation



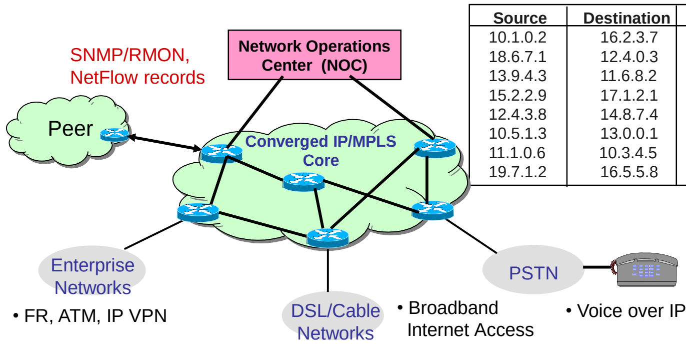
Motivating Examples: Financial Applications



Motivating Examples: Web Data Streams

- Mining query streams.
  - Google wants to know what queries are more frequent today than yesterday.
- Mining click streams.
  - Yahoo wants to know which of its pages are getting an unusual number of hits in the past hour.

Motivating Examples: Network Monitoring

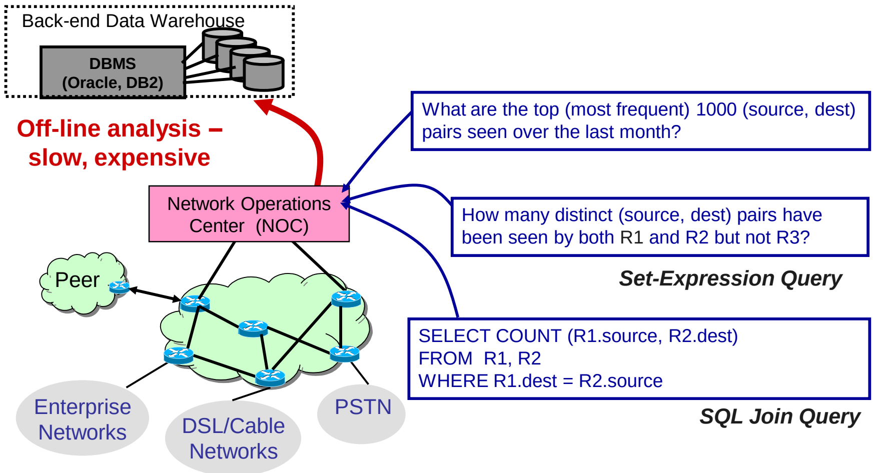


Source	Destination	Duration	Bytes	Protocol
10.1.0.2	16.2.3.7	12	20K	http
18.6.7.1	12.4.0.3	16	24K	http
13.9.4.3	11.6.8.2	15	20K	http
15.2.2.9	17.1.2.1	19	40K	http
12.4.3.8	14.8.7.4	26	58K	http
10.5.1.3	13.0.0.1	27	100K	ftp
11.1.0.6	10.3.4.5	32	300K	ftp
19.7.1.2	16.5.5.8	18	80K	ftp

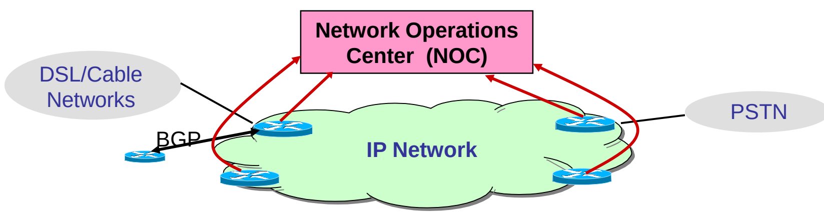
Example NetFlow IP Session Data

- 24x7 IP packet/flow data-streams at network elements
- Truly massive streams arriving at rapid rates
  - AT&T collects 600-800 Gigabytes of NetFlow data each day.
- Often shipped off-site to data warehouse for off-line analysis

Motivating Examples: Network Monitoring



Motivating Examples: Network Monitoring

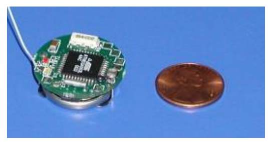
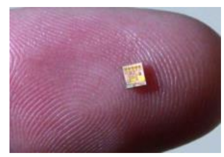


- Must process network streams in *real-time* and *one pass*
- Critical NM tasks: fraud, DoS attacks, SLA violations
  - Real-time traffic engineering to improve utilization
- Tradeoff communication and computation to reduce load
  - Make responses fast, minimize use of network resources
  - Secondarily, minimize space and processing cost at nodes

Motivating Examples: Sensor Networks

- the sensors era
  - ubiquitous, small, inexpensive sensors
  - applications that bridge physical world to information technology



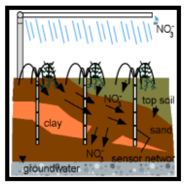



Motivating Examples: Sensor Networks

- the sensors era
  - ubiquitous, small, inexpensive sensors
  - applications that bridge physical world to information technology

Motivating Examples: Sensor Networks

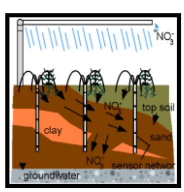
- the sensors era
  - ubiquitous, small, inexpensive sensors
  - applications that bridge physical world to information technology

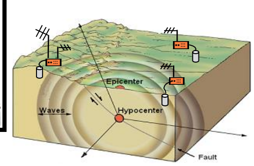


17

Motivating Examples: Sensor Networks

- the sensors era
  - ubiquitous, small, inexpensive sensors
  - applications that bridge physical world to information technology

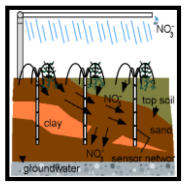


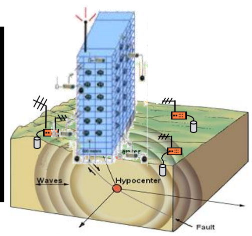


18

Motivating Examples: Sensor Networks

- the sensors era
  - ubiquitous, small, inexpensive sensors
  - applications that bridge physical world to information technology

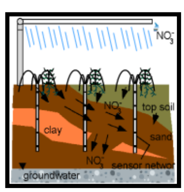


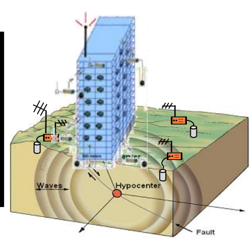


19

Motivating Examples: Sensor Networks

- the sensors era
  - ubiquitous, small, inexpensive sensors
  - applications that bridge physical world to information technology

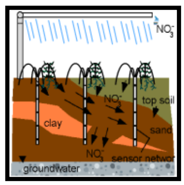


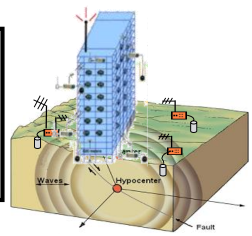


20

Motivating Examples: Sensor Networks

- the sensors era
  - ubiquitous, small, inexpensive sensors
  - applications that bridge physical world to information technology
- sensors unveil previously unobservable phenomena





Requirements

- develop efficient streaming algorithms
  - need to process this data online
  - allow approximate answers
  - operate in a distributed fashion (network as distributed database)
  - can also be used as one-pass algorithms for massive datasets

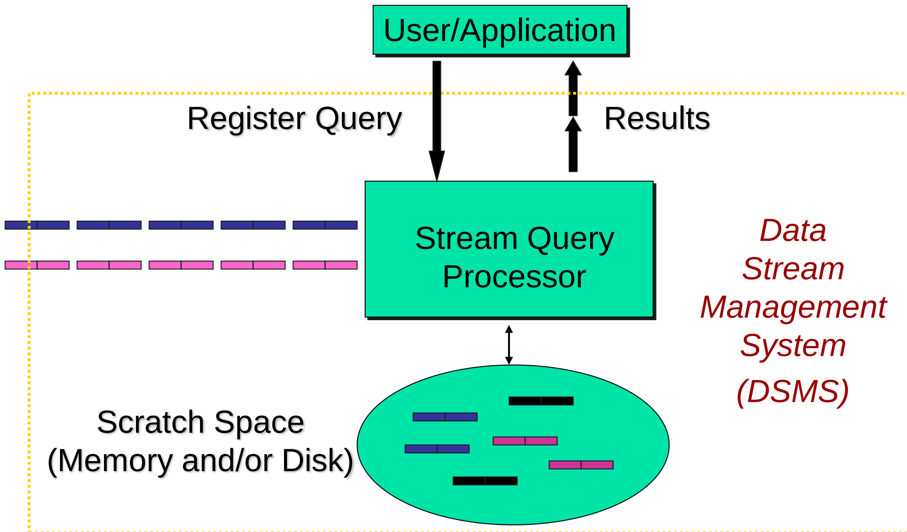
Requirements

- develop efficient streaming algorithms
  - need to process this data online
  - allow approximate answers
  - operate in a distributed fashion (network as distributed database)
  - can also be used as one-pass algorithms for massive datasets
- propose new data mining algorithms
  - help in data analysis in the above setting

Data Stream Management System?

- Traditional DBMS – data stored in finite, persistent data sets
- New Applications – data input as continuous, ordered data streams
  - Network monitoring and traffic engineering
  - Telecom call records
  - Network security
  - Financial applications
  - Sensor networks
  - Manufacturing processes
  - Web logs and clickstreams
  - Massive data sets

Data Stream Management System!



Meta-Questions

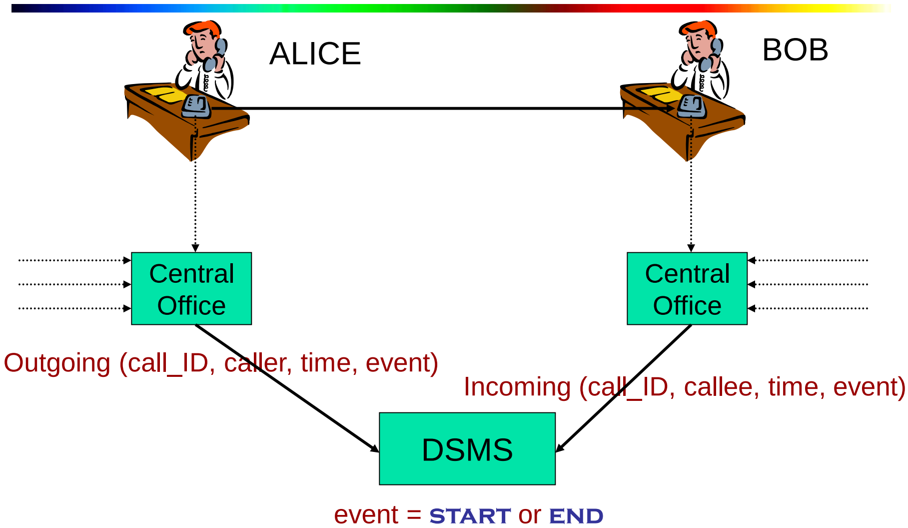
- Killer-apps
  - Application stream rates exceed DBMS capacity?
  - Can DSMS handle high rates anyway?
- Motivation
  - Need for general-purpose DSMS?
  - Not ad-hoc, application-specific systems?
- Non-Trivial
  - DSMS = merely DBMS with enhanced support for triggers, temporal constructs, data rate mgmt?

DBMS versus DSMS

- Persistent relations
- One-time queries
- Random access
- "Unbounded" disk store
- Only current state matters
- Passive repository
- Relatively low update rate
- No real-time services
- Precise answers
- Access plan determined by query processor, physical DB design

- Transient streams
- Continuous queries
- Sequential access
- Bounded main memory
- History/arrival-order is critical
- Active stores
- Possibly multi-GB arrival rate
- Real-time requirements
- Imprecise/approximate answers
- Access plan dependent on variable data arrival and data characteristics

Making Things Concrete



Query 1 (SELF-JOIN)

- Find all outgoing calls longer than 2 minutes

```
SELECT   O1.call_ID, O1.caller
FROM     Outgoing O1, Outgoing O2
WHERE    (O2.time – O1.time > 2
            AND   O1.call_ID = O2.call_ID
            AND   O1.event = START
          AND   O2.event = END)
```

- Result requires unbounded storage
- Can provide result as data stream
- Can output after 2 min, without seeing END

Query 2 (JOIN)

- Pair up callers and callees

```
SELECT   O.caller, I.callee
FROM     Outgoing O, Incoming I
WHERE    O.call_ID = I.call_ID
```

- Can still provide result as data stream
- Requires unbounded temporary storage …
- … unless streams are near-synchronized

Query 3 (group-by aggregation)

- Total connection time for each caller

```
SELECT      O1.caller, sum(O2.time – O1.time)
FROM        Outgoing O1, Outgoing O2
WHERE       (O1.call_ID = O2.call_ID
              AND O1.event = START
            AND  O2.event = END)
GROUP BY    O1.caller
```

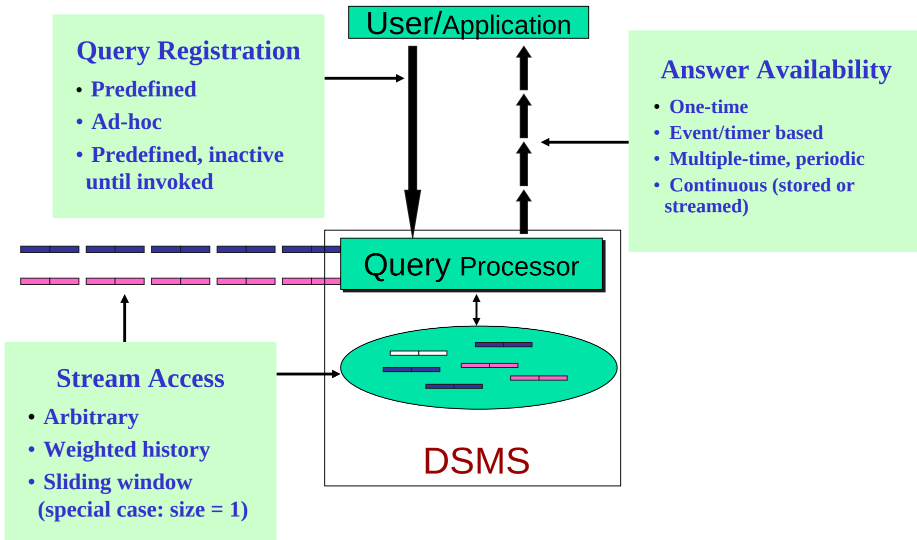
- Cannot provide result in (append-only) stream
  - Output updates?
  - Provide current value on demand?
  - Memory?

Data Model

- Append-only
  - Call records
- Updates
  - Stock tickers
- Deletes
  - Transactional data
- Meta-Data
  - Control signals, punctuations

System Internals – probably need all above

Query Model



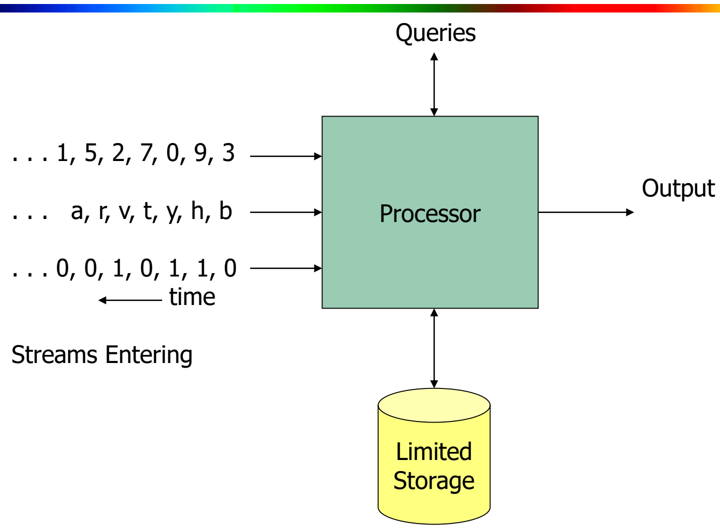
Related Database Technology

- DSMS must use ideas, but none is substitute
  - Triggers, Materialized Views in Conventional DBMS
  - Main-Memory Databases
  - Distributed Databases
  - Pub/Sub Systems
  - Active Databases
  - Sequence/Temporal/Timeseries Databases
  - Realtime Databases
  - Adaptive, Online, Partial Results
- Novelty in DSMS
  - Semantics: input ordering, streaming output, …
  - State: cannot store unending streams, yet need history
  - Performance: rate, variability, imprecision, …

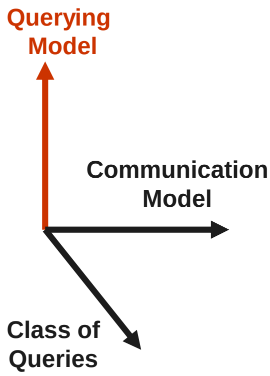
Stream Projects

- Amazon/Cougar (Cornell) – sensors
- Borealis (Brown/MIT) – sensor monitoring, dataflow
- Hancock (AT&T) – telecom streams
- Niagara (OGI/Wisconsin) – Internet XML databases
- OpenCQ (Georgia) – triggers, incr. view maintenance
- Stream (Stanford) – general-purpose DSMS
- Tapestry (Xerox) – pub/sub content-based filtering
- Telegraph (Berkeley) – adaptive engine for sensors
- Tribeca (Bellcore) – network monitoring

Is that all?



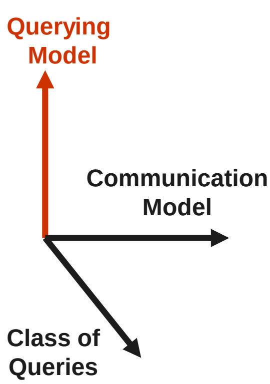
Distributed Stream Querying Space



"One-shot" vs. Continuous Querying

- One-shot queries: On-demand "pull" query answer from network
  - One or few rounds of communication
  - Nodes may prepare for a class of queries
- Continuous queries: *Track/monitor* answer at query site *at all times*
  - Detect anomalous/outlier behavior *in (near) real-time,* i.e., "Distributed triggers"
  - Challenge is to minimize communication Use "push-based" techniques
    May use one-shot algs as subroutines

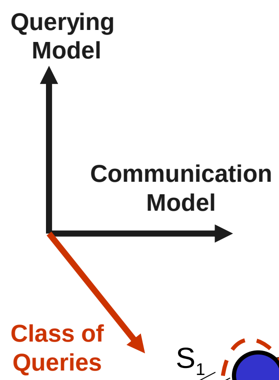
Distributed Stream Querying Space



Minimizing communication often needs approximation and randomization

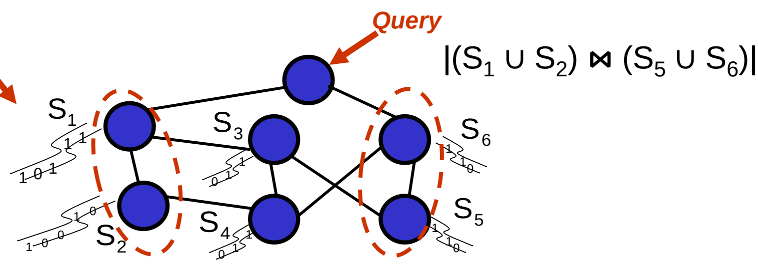
- E.g., Continuously monitor average value
  - Must send every change for exact answer
  - Only need 'significant' changes for approx (def. of "significant" specifies an algorithm)
- Probability sometimes vital to reduce communication
  - `count distinct` in one shot model needs randomness
  - Else **must** send complete data

Distributed Stream Querying Space

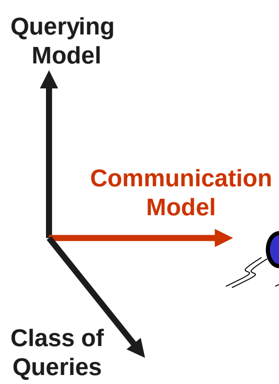


Class of Queries of Interest

- Simple algebraic vs. holistic aggregates
  - E.g., `count/max` vs. quantiles/top-k
- Duplicate-sensitive vs. duplicate-insensitive
  - "Bag" vs. "set" semantics
- Complex correlation queries
  - E.g., distributed joins, set expressions, …

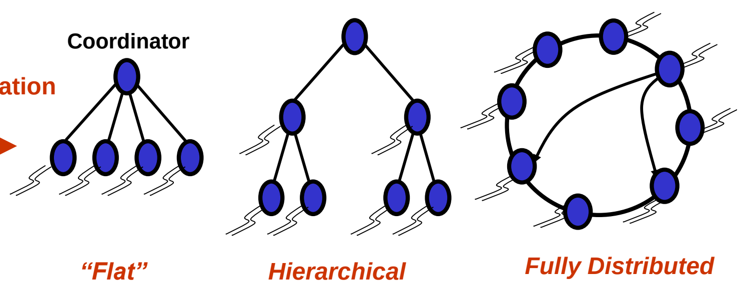


Distributed Stream Querying Space



Communication Network Characteristics

Topology: "Flat" vs. Hierarchical vs. Fully-distributed (e.g., P2P DHT)



Other network characteristics:

– Unicast (traditional wired), multicast, broadcast (radio nets)

– Node failures, loss, intermittent connectivity, …

Unrestricted Window

- One model of stream processing is when queries refer to all the data in a *window* that starts at the "beginning of time", extends up to the current time, and continuous expanding with time (potentially infinite length).

Unrestricted Window

$\boxed{q}$ w e r t y u i o p a s d f g h j k l z x c v b n m

← Past Future →

Unrestricted Window

q w e r t y u i o p a s d f g h j k l z x c v b n m

q w e r t y u i o p a s d f g h j k l z x c v b n m

← Past Future →

Unrestricted Window

q w e r t y u i o p a s d f g h j k l z x c v b n m

q w e r t y u i o p a s d f g h j k l z x c v b n m

q w e r t y u i o p a s d f g h j k l z x c v b n m

← Past Future →

Unrestricted Window

q w e r t y u i o p a s d f g h j k l z x c v b n m

q w e r t y u i o p a s d f g h j k l z x c v b n m

q w e r t y u i o p a s d f g h j k l z x c v b n m

...

q w e r t y u i o p a s d f g h j k l z x c v b n m

← Past Future →

Unrestricted Window

- One model of stream processing is when queries refer to all the data in a *window* that starts at the "beginning of time", extends up to the current time, and continuous expanding with time (potentially infinite length).
- What happens when we try to compute joins in this model?

Unrestricted Window

- One model of stream processing is when queries refer to all the data in a *window* that starts at the "beginning of time", extends up to the current time, and continuous expanding with time (potentially infinite length).
- What happens when we try to compute joins in this model?
  - Join results involving some piece of data may appear at any time in the future

Unrestricted Window

- One model of stream processing is when queries refer to all the data in a *window* that starts at the "beginning of time", extends up to the current time, and continuous expanding with time (potentially infinite length).
- What happens when we try to compute joins in this model?
  - Join results involving some piece of data may appear at any time in the future
  - In order to correctly compute the result, we need to store all values that have appeared in the past!

Shifting Window

- Another model of stream processing is that queries are about a *window* of length *N*, and this window advances by *N,* where *N* are the most recent elements received, or the most recent time units.

Shifting Window

q w e r t [y u i o p] a s d f g h j k l z x c v b n m

← Past Future →

Shifting Window

q w e r t y u i o p a s d f g h j k l z x c v b n m

q w e r t y u i o p a s d f g h j k l z x c v b n m

← Past Future →

Shifting Window

q w e r t y u i o p a s d f g h j k l z x c v b n m

q w e r t y u i o p a s d f g h j k l z x c v b n m

q w e r t y u i o p a s d f g h j k l z x c v b n m

← Past Future →

Shifting Window

q w e r t y u i o p a s d f g h j k l z x c v b n m

q w e r t y u i o p a s d f g h j k l z x c v b n m

q w e r t y u i o p a s d f g h j k l z x c v b n m

q w e r t y u i o p a s d f g h j k l z x c v b n m

← Past Future →

Shifting Window

- Another model of stream processing is that queries are about a *window* of length *N*, and this window advances by *N*, where *N* are the most recent elements received, or the most recent time units.

- Useful queries within this model:
  - average number of calls every day
  - std deviation of packet losses every 10 minutes
  - etc.

Sliding Window

- A useful model of stream processing is that queries are about a *window* of length N, where N are the most recent elements received, or the most recent time units.

Sliding Window

q w e r t y u i o [p a s d f] g h j k l z x c v b n m

q w e r t y u i o p [a s d f g] h j k l z x c v b n m

q w e r t y u i o p a [s d f g h] j k l z x c v b n m

q w e r t y u i o p a s [d f g h j] k l z x c v b n m

← Past Future →

Sliding Window

- A useful model of stream processing is that queries are about a *window* of length N, where N are the most recent elements received, or the most recent time units.
- Interesting case: N is so large it cannot be stored in memory, or even on disk.
  - Or, there are so many streams that we cannot store the values for all the windows.

Counting Bits --- (1)

- Problem: given a stream of 0's and 1's, be prepared to answer queries of the form "how many 1's in the last k bits?" where $k \leq N$.
- Obvious solution: store the most recent N bits.
  - When new bit comes in, discard the $N+1^{st}$ bit.

Counting Bits --- (2)

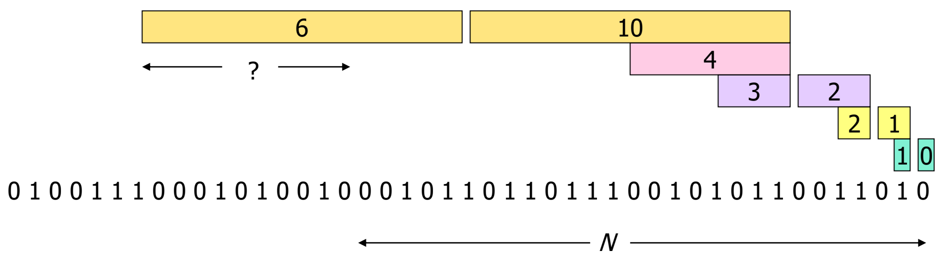
- You can't get an exact answer without storing the entire window.
- Real Problem: what if we cannot afford to store N bits?
  - E.g., we are processing 1 trillion streams and N = 1 trillion, but we're happy with an approximate answer.

Something That Doesn't (Quite) Work

- Summarize exponentially increasing regions of the stream, looking backward.
- Drop small regions if they begin at the same point as a larger region.

Example

We can construct the count of the last N bits, except we're Not sure how many of the last 6 are included.



What's Good?

- Stores only $O(\log^2 N)$ bits.
  - $O(\log N)$ counts of $\log_2 N$ bits each.
- Easy update as more bits enter.
- Error in count no greater than the number of 1's in the "unknown" area.

What's Not So Good?

- As long as the 1's are fairly evenly distributed, the error due to the unknown region is small --- no more than 50%.
- But it could be that all the 1's are in the unknown area at the end.
- In that case, the error is unbounded.

Fixup

- Instead of summarizing fixed-length blocks, summarize blocks with specific numbers of 1's.
  - Let the block "sizes" (number of 1's) increase exponentially.
- When there are few 1's in the window, block sizes stay small, so errors are small.

DGIM* Method

- Store $O(\log^2 N)$ bits per stream.
- Gives approximate answer, never off by more than 50%.
  - Error factor can be reduced to any fraction > 0, with more complicated algorithm and proportionally more stored bits.

*Datar, Gionis, Indyk, and Motwani

Timestamps

- Each bit in the stream has a *timestamp*, starting 1, 2, …
- Record timestamps modulo N (the window size), so we can represent any relevant timestamp in $O(\log_2 N)$ bits.

Buckets

- A *bucket* in the DGIM method is a record consisting of:
  1. The timestamp of its end [O(log N) bits].
  2. The number of 1's between its beginning and end [O(log log N) bits].

- Constraint on buckets: number of 1's must be a power of 2.
  - That explains the log log N in (2).

Representing a Stream by Buckets

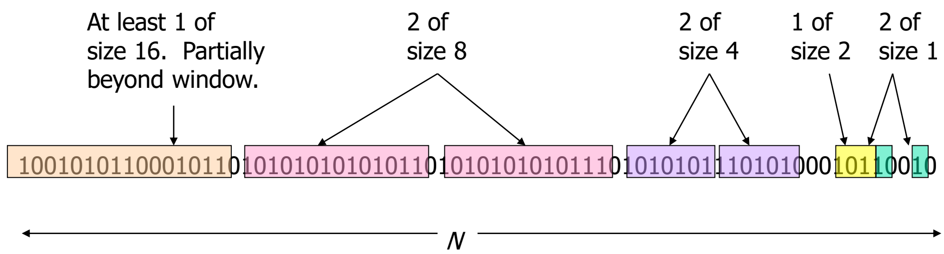
- Either one or two buckets with the same power-of-2 number of 1's.

- Buckets do not overlap in timestamps.

- Buckets are sorted by *size* (# of 1's).
  - Earlier buckets are not smaller than later buckets.

- Buckets disappear when their end-time is > N time units in the past.

Example



Updating Buckets --- (1)

- When a new bit comes in, drop the last (oldest) bucket if its end-time is prior to N time units before the current time.
- If the current bit is 0, no other changes are needed.

Updating Buckets --- (2)

- If the current bit is 1:
  1. Create a new bucket of size 1, for just this bit.
     - End timestamp = current time.
  2. If there are now three buckets of size 1, combine the oldest two into a bucket of size 2.
  3. If there are now three buckets of size 2, combine the oldest two into a bucket of size 4.
  4. And so on…

Example

1001010110001011 0 101010101011 0 101010101011 0 1010101 110101 000 101 1 00 1 0

Example

10010101100010110101010101010110101010101011101010101110101000101100 10

0010101100010110101010101010110101010101011101010101110101000101100101

Example

10010101100010110101010101010110101010101011101010101110101000101100 10

0010101100010110101010101010110101010101011101010101110101000101100101

Example

1001010110001011010101010101011010101010101110101010111010100010110010

0010101100010110101010101010110101010101011101010101110101000101100101

Example

1001010110001011010101010101011010101010101110101010111010100010110010

0010101100010110101010101010110101010101011101010101110101000101100101

0010101100010110101010101010110101010101011101010101110101000101100101

Example

10010101100010110101010101010110101010101011101010101110101000 10110010

0010101100010110101010101010110101010101011101010101110101000101100101

0010101100010110101010101010110101010101011101010101110101000101100101

0101100010110101010101010110101010101011101010101110101000101100101101

Example

10010101100010110101010101010110101010101011101010101110101000 10110010

0010101100010110101010101010110101010101011101010101110101000101100101

0010101100010110101010101010110101010101011101010101110101000101100101

0101100010110101010101010110101010101011101010101110101000101100101101

0101100010110101010101010110101010101011101010101110101000101100101101

Example

10010101100010110101010101010110101010101011101010101110101000101110010

00101011000101101010101010101101010101010111010101011101010001011001 01

00101011000101101010101010101101010101010111010101011101010001011001 01

0101100010110101010101010101101010101010111010101011101010001011001 01 101

0101100010110101010101010101101010101010111010101011101010001011001 01101

Example

10010101100010110101010101010110101010101011101010101110101000101110010

00101011000101101010101010101101010101010111010101011101010001011001 01

00101011000101101010101010101101010101010111010101011101010001011001 01

0101100010110101010101010101101010101010111010101011101010001011001 01 101

0101100010110101010101010101101010101010111010101011101010001011001 01101

Example

1001010110001011010101010101011010101010101110101010111010100010110010

0010101100010110101010101010110101010101011101010101110101000101100101

0010101100010110101010101010110101010101011101010101110101000101100101

0101100010110101010101010110101010101011101010101110101000101100101101

0101100010110101010101010110101010101011101010101110101000101100101101

Example

1001010110001011010101010101011010101010101110101010111010100010110010

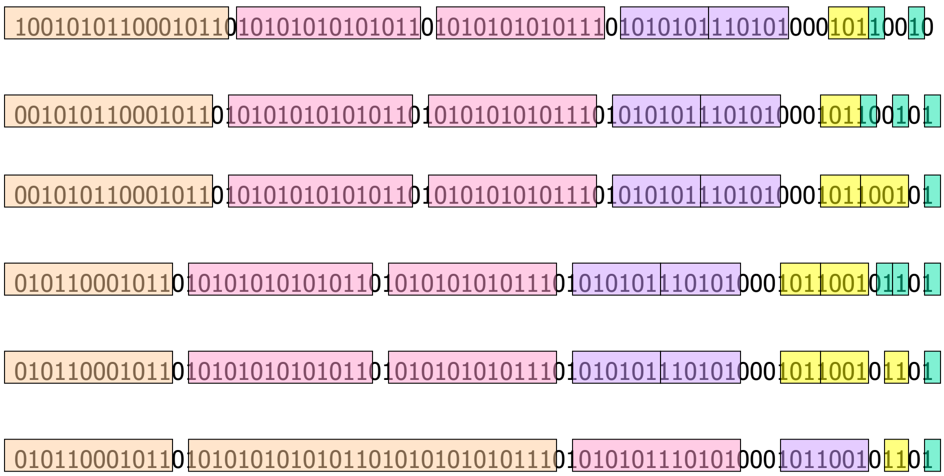
0010101100010110101010101010110101010101011101010101110101000101100101

0010101100010110101010101010110101010101011101010101110101000101100101

0101100010110101010101010110101010101011101010101110101000101100101101

0101100010110101010101010110101010101011101010101110101000101100101101

Example



Querying

- To estimate the number of 1's in the most recent *N* bits:
  1. Sum the sizes of all buckets but the last.
  2. Add in half the size of the last bucket.
- Remember, we don't know how many 1's of the last bucket are still within the window.

Error Bound

- Suppose the last bucket has size 2^k.
- Then by assuming 2^{k-1} of its 1's are still within the window, we make an error of at most 2^{k-1}.
- Since there is at least one bucket of each of the sizes less than 2^k, the true sum is no less than 2^k-1.
- Thus, error at most 50%.

More Stream Mining

- Counting Distinct Elements
- Computing "Moments"
- Frequent Itemsets
- Elephants and Troops
- Exponentially Decaying Windows

Counting Distinct Elements

- Problem: a data stream consists of elements chosen from a set of size *n*. Maintain a count of the number of distinct elements seen so far.
- Obvious approach: maintain the set of elements seen.

Applications

- How many different words are found among the Web pages being crawled at a site?
  - Unusually low or high numbers could indicate artificial pages (spam?).
- How many different Web pages does each customer request in a week?

Using Small Storage

- Real Problem: what if we do not have space to store the complete set?
- Estimate the count in an unbiased way.
- Accept that the count may be in error, but limit the probability that the error is large.

Flajolet-Martin* Approach

- 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 0's in $h(a)$.
- Record R = the maximum $r(a)$ seen.
- Estimate = 2^R.

* Really based on a variant due to AMS (Alon, Matias, and Szegedy)

Why It Works

- The probability that a given $h(a)$ ends in at least r 0's is 2^{-r}.
- If there are m different elements, the probability that $R \geq r$ is $1 - \boxed{(1 - \boxed{2^{-r}})^m}$.

Prob. all h(a)'s end in fewer than r 0's.

Prob. a given h(a) ends in fewer than r 0's.

Why It Works – (2)

- Since 2^{-r} is small, $1 - (1-2^{-r})^m \approx 1 - e^{-m2^{-r}}$.
- If $2^r >> m$, $1 - (1 - 2^{-r})^m \approx 1 - (1 - m2^{-r}) \approx m/2^r \approx 0$.

First 2 terms of the Taylor expansion of e^x

- If $2^r << m$, $1 - (1 - 2^{-r})^m \approx 1 - e^{-m2^{-r}} \approx 1$.
- Thus, 2^R will almost always be around m.

Why It Doesn't Work

- $E(2^R)$ is actually infinite.
  - Probability halves when R -> R+1, but value doubles.
- Workaround involves using many hash functions and getting many samples.
- How are samples combined?
  - Average? What if one very large value?
  - Median? All values are a power of 2.

Solution

- Partition your samples into small groups.
- Take the average of groups.
- Then take the median of the averages.

Generalization: Moments

- Suppose a stream has elements chosen from a set of n values.
- Let m_i be the number of times value i occurs.
- The kth *moment* is the sum of $(m_i)^k$ over all i.

Special Cases

- 0th moment = number of different elements in the stream.
  - The problem just considered.
- 1st moment = count of the numbers of elements = length of the stream.
  - Easy to compute.
- 2nd moment = *surprise number* = a measure of how uneven the distribution is.

Example: Surprise Number

- Stream of length 100; 11 values appear.
- Unsurprising: 10, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9. Surprise # = 910.
- Surprising: 90, 1, 1, 1, 1, 1, 1, 1 ,1, 1, 1. Surprise # = 8,110.

AMS Method

- Works for all moments; gives an unbiased estimate.
- We'll just concentrate on 2nd moment.
- Based on calculation of many random variables *X*.
  - Each requires a count in main memory, so number is limited.

One Random Variable

- Assume stream has length n.
- Pick a random time to start, so that any time is equally likely.
- Let the chosen time have element a in the stream.
- $X = n * ((\text{twice the number of } a\text{'s in the stream starting at the chosen time}) - 1)$.
  - Note: store n once, count of a's for each X.

Expected Value of *X*

- 2nd moment is $\Sigma_a (m_a)^2$.
- $E(X) = (1/n)\big(\Sigma_{\text{all times } t}\ n * (\text{twice the number of times the stream element at time } t \text{ appears from that time on}) - 1\big)$.
- $= \Sigma_a (1/n)(n)(1+3+5+\ldots+2m_a-1)$.
- $= \Sigma_a (m_a)^2$.

Group times by the value seen

Time when the last a is seen

Time when the penultimate a is seen

Time when the first a is seen

Combining Samples

- Compute as many variables *X* as can fit in available memory.
- Average them in groups.
- Take median of averages.
- Proper balance of group sizes and number of groups assures not only correct expected value, but expected error goes to 0 as number of samples gets large.

Problem: Streams Never End

- We assumed there was a number *n*, the number of positions in the stream.
- But real streams go on forever, so *n* is a variable – the number of inputs seen so far.

Fixups

1. The variables X have n as a factor – keep n separately; just hold the count in X.
2. Suppose we can only store k counts. We must throw some X's out as time goes on.
   - Objective: each starting time t is selected with probability k/n.

Solution to (2)

- Choose the first k times for k variables.
- When the n^{th} element arrives ($n > k$), choose it with probability k / n.
- If you choose it, throw one of the previously stored variables out, with equal probability.

New Topic: Counting Itemsets

- Problem: given a stream, which items appear more than s times in the window?
- Possible solution: think of the stream of baskets as one binary stream per item.
  - 1 = item present; 0 = not present.
  - Use DGIM to estimate counts of 1's for all items.

Extensions

- In principle, you could count frequent pairs or even larger sets the same way.
  - One stream per itemset.
- Drawbacks:
  1. Only approximate.
  2. Number of itemsets is way too big.

Approaches

1. "Elephants and troops": a heuristic way to converge on unusually strongly connected itemsets.
2. Exponentially decaying windows: a heuristic for selecting likely frequent itemsets.

Elephants and Troops

- When Sergey Brin wasn't worrying about Google, he tried the following experiment.
- Goal: find unusually correlated sets of words.
  - "*High Correlation*" = frequency of occurrence of set >> product of frequencies of members.

Experimental Setup

- The data was an early Google crawl of the Stanford Web.
- Each night, the data would be streamed to a process that counted a preselected collection of itemsets.
  - If {*a, b, c*} is selected, count {*a, b, c*}, {*a*}, {*b*}, and {*c*}.
  - "Correlation" = $n^2 \times$#abc/(#a$\times$#b$\times$#c).
    - *n* = number of pages.

After Each Night's Processing . . .

1. Find the most correlated sets counted.
2. Construct a new collection of itemsets to count the next night.
   - All the most correlated sets ("*winners*").
   - Pairs of a word in some winner and a random word.
   - Winners combined in various ways.
   - Some random pairs.

After a Week . . .

- The pair {"elephants", "troops"} came up as the big winner.
- Why? It turns out that Stanford students were playing a Punic-War simulation game internationally, where moves were sent by Web pages.

New Topic: Mining Streams Versus Mining DB's

- Unlike mining databases, mining streams doesn't have a fixed answer.
- We're really mining in the "Stat" point of view, e.g., "Which itemsets are frequent in the underlying model that generates the stream?"

Stationarity

Our assumptions make a big difference:

1. Is the model *stationary* ?
   - I.e., are the same statistics used throughout all time to generate the stream?
2. Or does the frequency of generating given items or itemsets change over time?

Some Options for Frequent Itemsets

1. Run periodic experiments, like E&T.
   - Like SON – itemset is a candidate if it is found frequent on any "day."
   - Good for stationary statistics.
2. Frame the problem as finding all frequent itemsets in an "exponentially decaying window."
   - Good for nonstationary statistics.

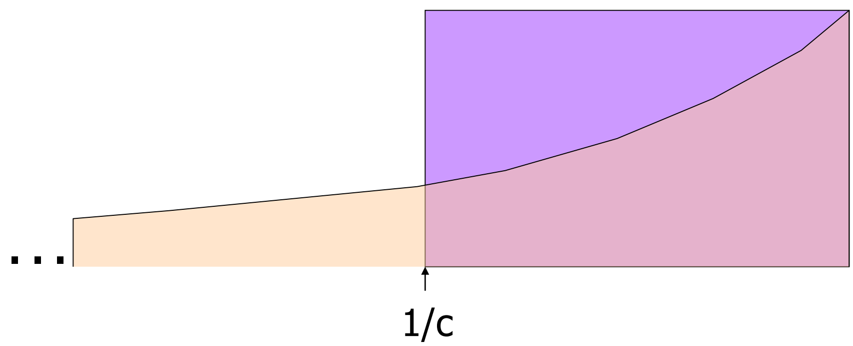
Exponentially Decaying Windows

- If stream is $a_1, a_2, \ldots$ and we are taking the sum of the stream, take the answer at time t to be: $\Sigma_{i = 1,2,\ldots,t}\, a_i\, e^{-c(t-i)}$.
- c is a constant, presumably tiny, like 10^{-6} or 10^{-9}.

Example: Counting Items

- If each a_i is an "item" we can compute the *characteristic function* of each possible item x as an E.D.W.
- That is: $\Sigma_{i = 1,2,\ldots,t}\, \delta_i\, e^{-c(t-i)}$, where $\delta_i = 1$ if $a_i = x$, and 0 otherwise.
  - Call this sum the "*count*" of item x.

Sliding Versus Decaying Windows



Counting Items – (2)

- Suppose we want to find those items of weight at least ½.
- Important property: sum over all weights is $1/(1 - e^{-c})$ or very close to $1/[1 - (1 - c)] = 1/c$.
- Thus: at most $2/c$ items have weight at least ½.

Extension to Larger Itemsets*

- Count (some) itemsets in an E.D.W.
- When a basket B comes in:
  1. Multiply all counts by $(1-c)$;
  2. For uncounted items in B, create new count.
  3. Add 1 to count of any item in B and to any counted itemset contained in B.
  4. Drop counts < ½.
  5. Initiate new counts (next slide).

* Informal proposal of Art Owen

Initiation of New Counts

- Start a count for an itemset $S \subseteq B$ if every proper subset of S had a count prior to arrival of basket B.
- Example: Start counting $\{i, j\}$ iff both i and j were counted prior to seeing B.
- Example: Start counting $\{i, j, k\}$ iff $\{i, j\}$, $\{i, k\}$, and $\{j, k\}$ were all counted prior to seeing B.

How Many Counts?

- Counts for single items $\leq$ ($2/c$) times the average number of items in a basket.
- Counts for larger itemsets = ??. But we are conservative about starting counts of large sets.
  - If we counted every set we saw, one basket of 20 items would initiate 1M counts.