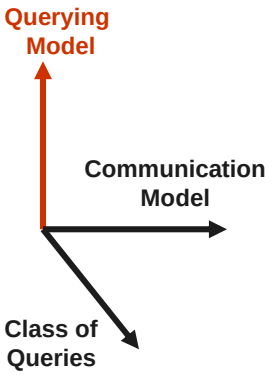
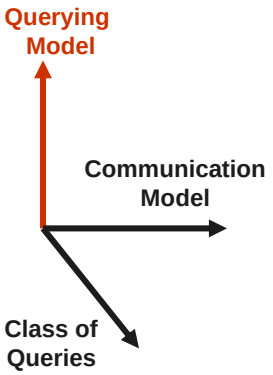
Distributed Stream Querying Space



"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



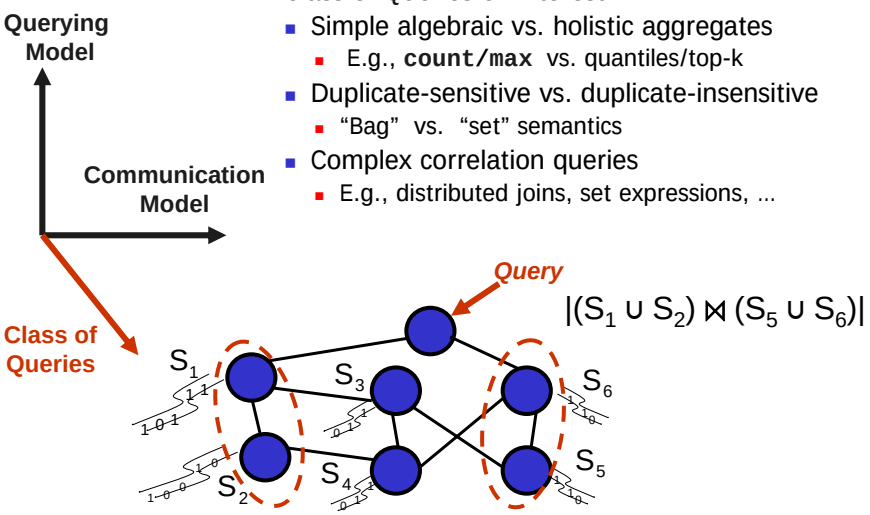
Minimizing communication often needs approximation and randomization

- 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

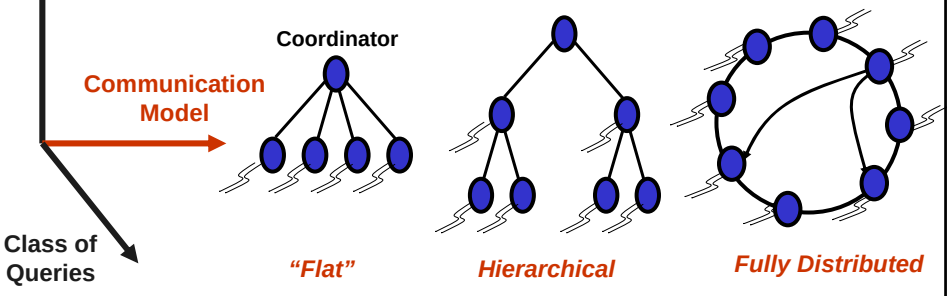
- Simple algebraic vs. holistic aggregates
  - E.g., **`count/max`** vs. quantiles/top-k
- Duplicate-sensitive vs. duplicate-insensitive
  - "Bag" vs. "set" semantics
- Complex correlation queries
  - E.g., distributed joins, set expressions, ...



Distributed Stream Querying Space

Communication Network Characteristics

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



Other network characteristics:

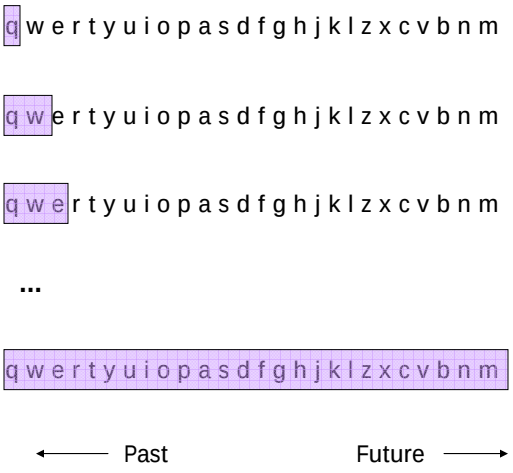
– Unicast (traditional wired), multicast, broadcast (radio nets)

– 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



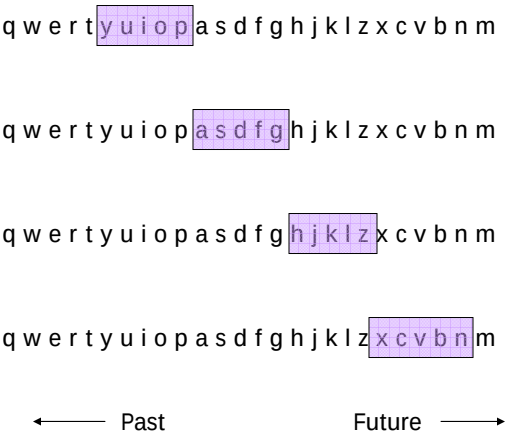
Unrestricted Window

- 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



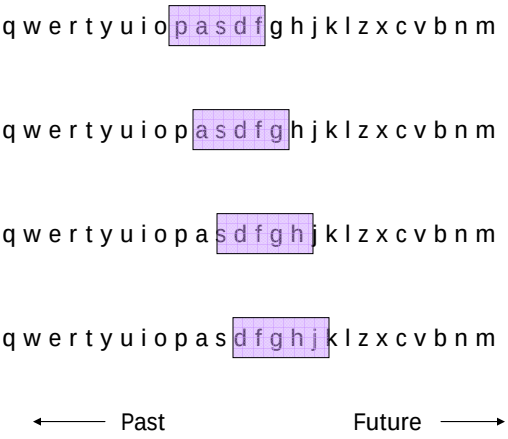
Shifting Window

- 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.
- Interesting case: *N* is so large it cannot be stored in memory, or even on disk.
  - Or, there are so many streams that windows for all cannot be stored.

Sliding Window



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)

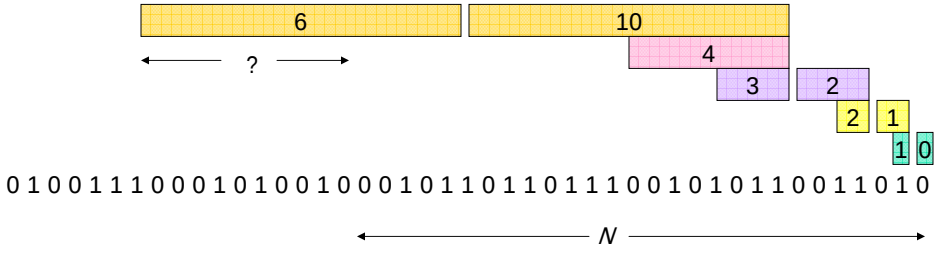
- You can't get an exact answer without storing the entire window.
- Real Problem: what if we cannot afford to store N bits?
  - E.g., we are processing 1 trillion streams and N = 1 trillion, but we're happy with an approximate answer.

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.

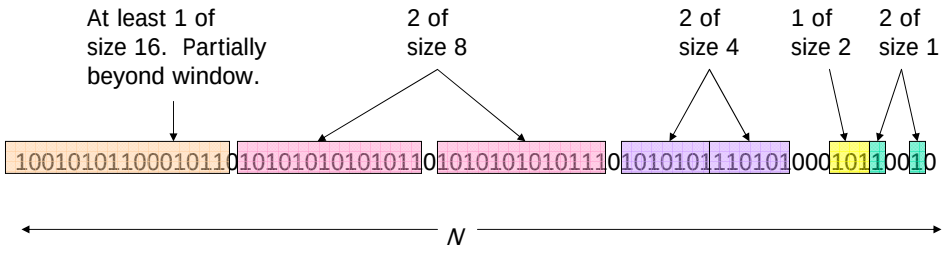
Buckets

- 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



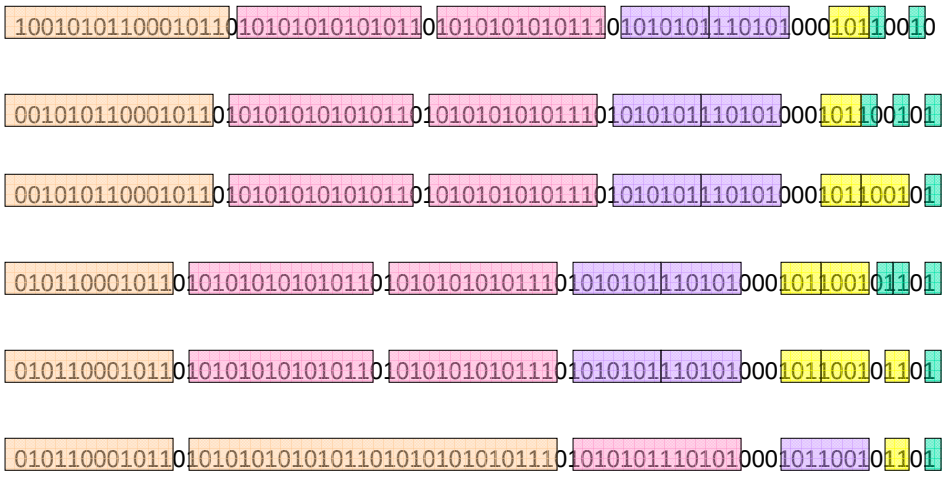
Updating Buckets --- (1)

- 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



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%.