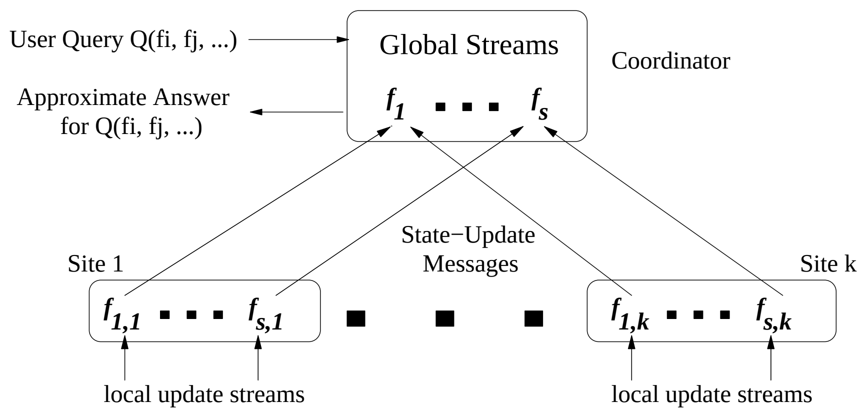
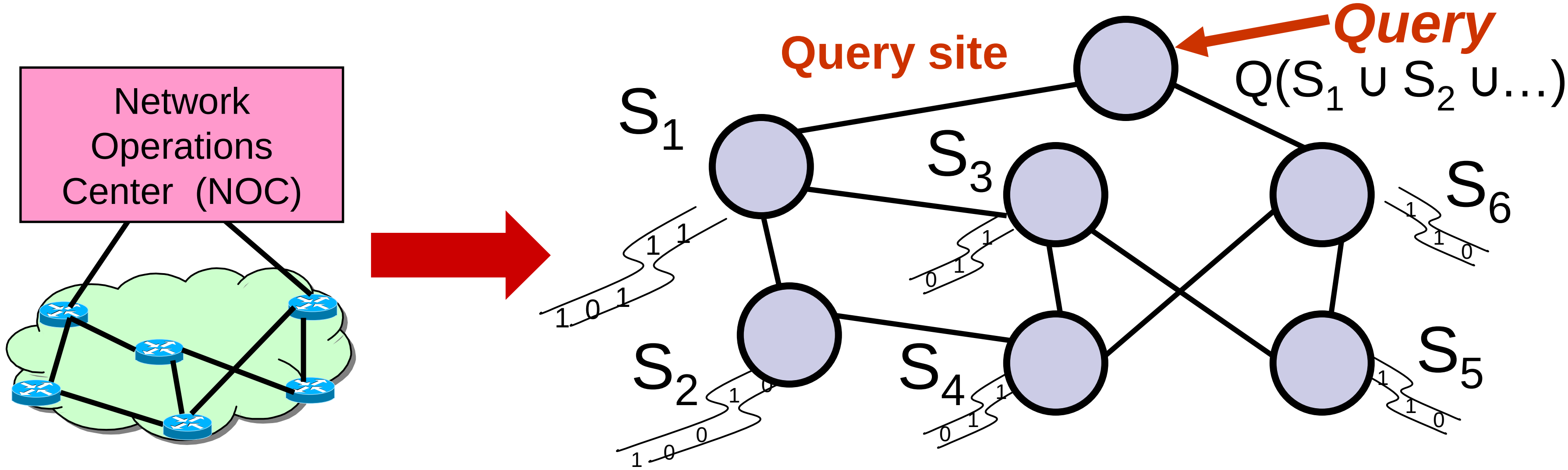
Managing *Distributed* Data Streams – II



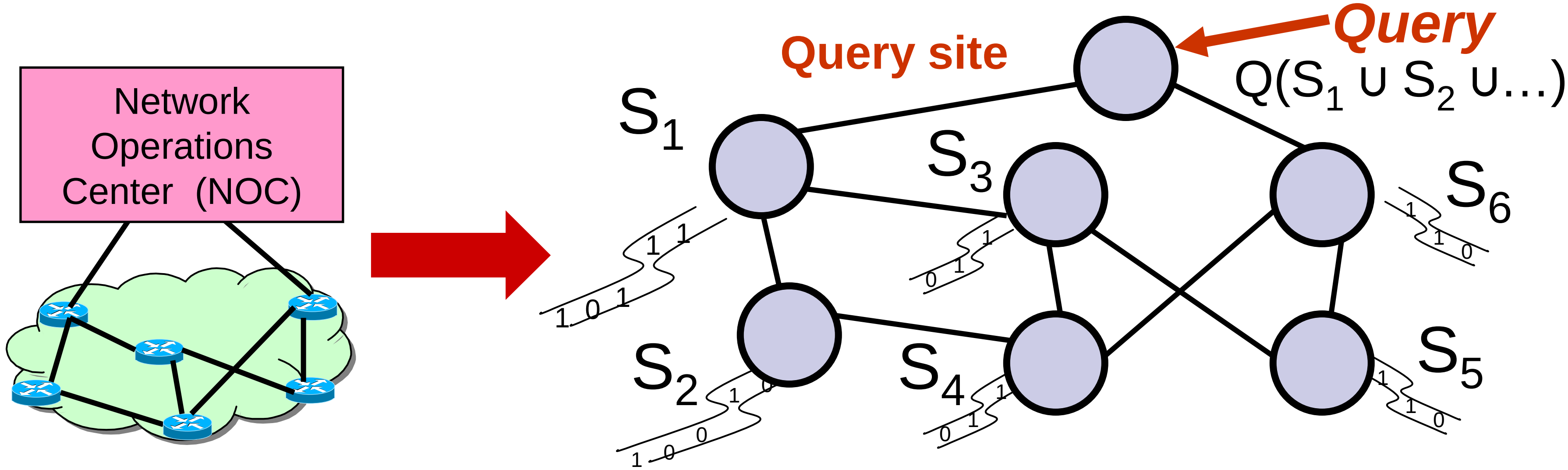
Slides based on the Cormode/Garofalakis VLDB'2006 tutorial

Distributed Streams Model



- Large-scale querying/monitoring: *Inherently distributed!*
  - Streams physically distributed across remote sites
    E.g., stream of UDP packets through subset of edge routers
- *Challenge is "holistic" querying/monitoring*
  - Queries over the *union of distributed streams* $Q(S_1 \cup S_2 \cup \ldots)$
  - Streaming data is spread throughout the network

Distributed Streams Model

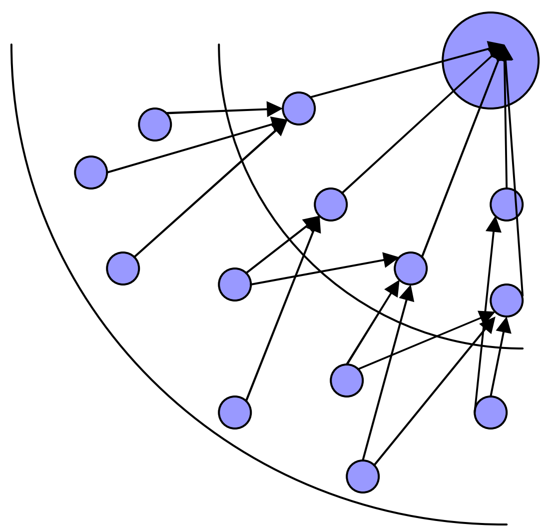


- Need timely, accurate, and efficient query answers
- Additional complexity over centralized data streaming!
- Need space/time- *and communication-efficient* solutions
  - Minimize network overhead
  - Maximize network lifetime (e.g., sensor battery life)
  - Cannot afford to “centralize” all streaming data

Outline

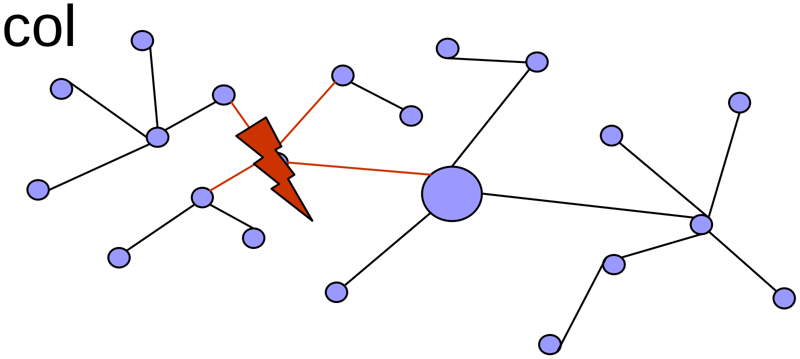
- Introduction, Motivation, Problem Setup
- One-Shot Distributed-Stream Querying
  - Tree Based Aggregation
  - Robustness and Loss
  - *Decentralized Computation and Gossiping*
- Continuous Distributed-Stream Tracking
- *Probabilistic Distributed Data Acquisition*
- Conclusions

Robustness and Loss



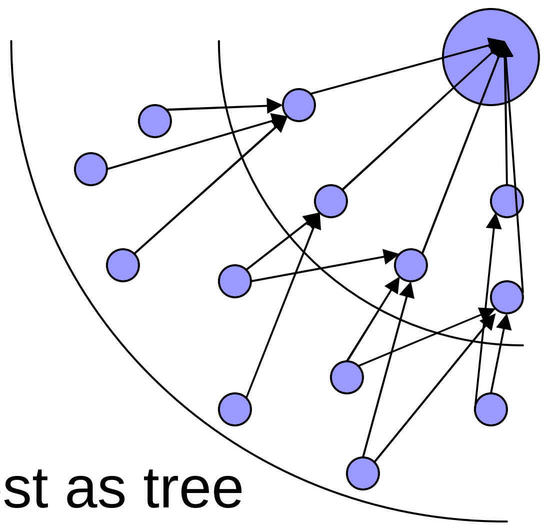
Unreliability

- Tree aggregation techniques assumed a reliable network
  - we assumed no node failure, nor loss of any message
- Failure can dramatically affect the computation
  - E.g., **`sum`** – if a node near the root fails, then a whole subtree may be lost
- Clearly a particular problem in sensor networks
  - If messages are lost, maybe can detect and resend
  - If a node fails, may need to rebuild the whole tree and re-run protocol
  - Need to detect the failure, could cause high uncertainty



Sensor Network Issues

- Sensor nets typically based on radio communication
  - So broadcast (within range) cost the same as unicast
  - Use multi-path routing: improved reliability, reduced impact of failures, less need to repeat messages
- E.g., computation of `max`
  - structure network into rings of nodes in equal hop count from root
  - listen to all messages from ring below, then send max of all values heard
  - converges quickly, high path diversity
  - each node sends only once, so same cost as tree

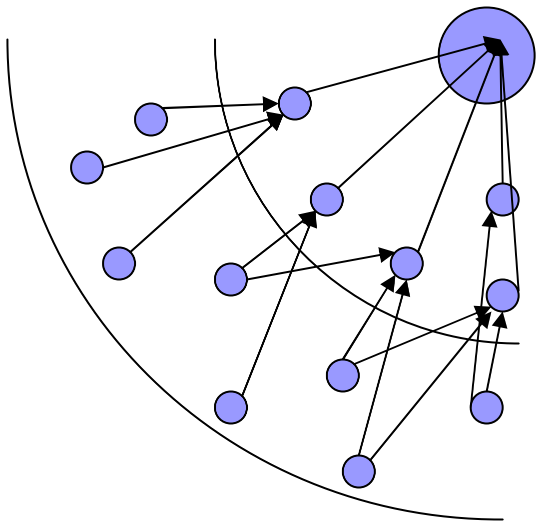


Order and Duplicate Insensitivity

- It works because **`max`** is Order and Duplicate Insensitive (ODI) [Nath et al.'04]
- Make use of the same e(), f(), g() framework as before
- Can prove correct if e(), f(), g() satisfy properties:
  - g gives same output for duplicates: $i=j \Rightarrow g(i) = g(j)$
  - f is associative and commutative:
    $f(x,y) = f(y,x); f(x,f(y,z)) = f(f(x,y),z)$
  - f is *same-synopsis idempotent*: $f(x,x) = x$
- Easy to check **`min`**, **`max`** satisfy these requirements, **`sum`** does not

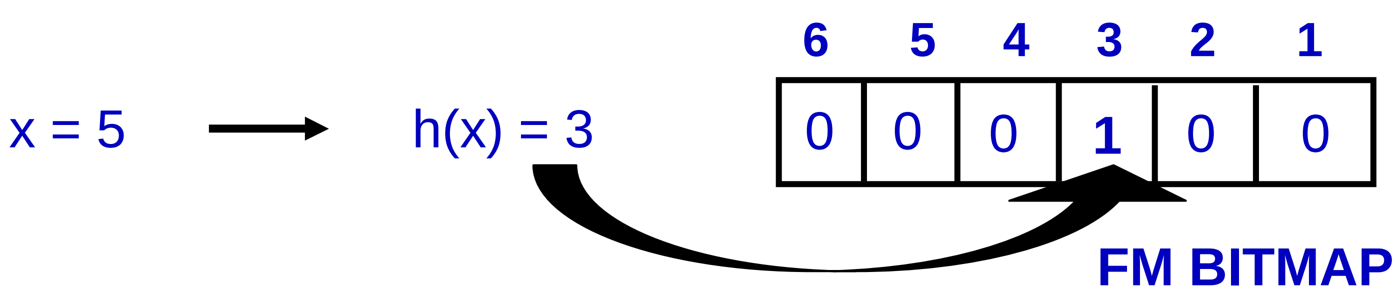
Applying ODI idea

- Only `max` and `min` seem to be “naturally” ODI
- How to make ODI summaries for other aggregates?
- Will make use of duplicate insensitive primitives:
  - Flajolet-Martin Sketch (FM)
  - Min-wise hashing
  - Random labeling
  - Bloom Filter



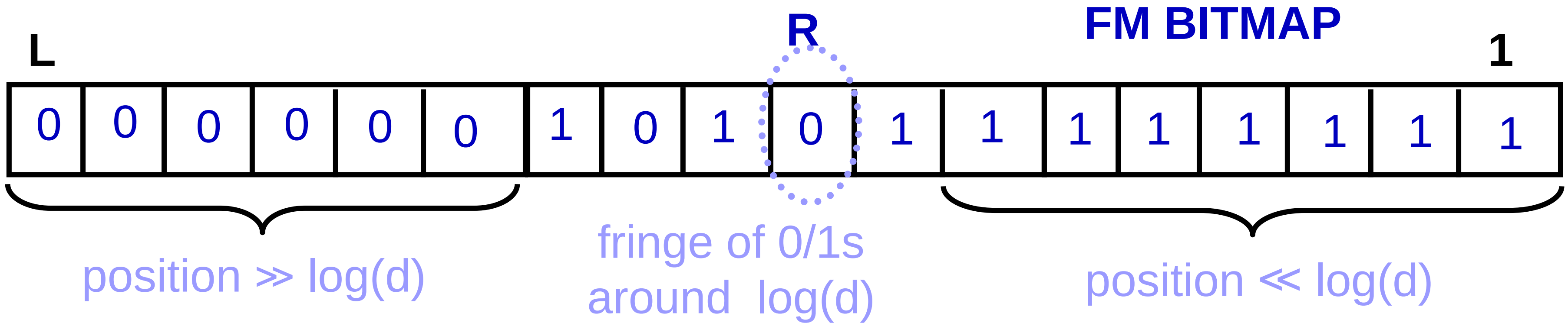
FM Sketch

- Estimates number of distinct inputs (**`count distinct`**)
- Uses hash function mapping input items to i with prob 2^{-i}
  - i.e. $Pr[h(x) = 1] = 1/2$, $Pr[h(x) = 2] = 1/4$, $Pr[h(x)=3] = 1/8$ …
  - Easy to construct h() from a uniform hash function by counting trailing zeros
- Maintain FM Sketch = bitmap array of $L = \log U$ bits
  - Initialize bitmap to all 0s
  - For each incoming value x, set FM[h(x)] = 1



FM Analysis

- If d distinct values, expect d/2 map to FM[1], d/4 to FM[2]…



  - Let R = position of rightmost zero in FM, indicator of log(d)
  - Basic estimate $d = c2^R$ for scaling constant $c \approx 1.3$
  - Average many copies (different hash fns) improves accuracy

FM Sketch – ODI Properties

6	5	4	3	2	1
0	0	1	0	1	1

+

6	5	4	3	2	1
0	1	1	0	0	1

=

6	5	4	3	2	1
0	1	1	0	1	1

- Fits into the Generate, Fuse, Evaluate framework.
  - Can fuse multiple FM summaries (with same hash h()): take bitwise-OR of the summaries
- With $O(1/\varepsilon^2 \log 1/\delta)$ copies, get $(1\pm\varepsilon)$ accuracy with probability at least $1-\delta$
  - 10 copies gets ≈ 30% error, 100 copies < 10% error
  - Can pack FM into eg. 32 bits. Assume h() is known to all.

FM within ODI

- What if we want to count, not count distinct?
  - E.g., each site i has a count c_i, we want $\sum_i c_i$
  - Tag each item with site ID, write in unary: $(i,1), (i,2)\ldots (i,c_i)$
  - Run FM on the modified input, and run ODI protocol
- What if counts are large?
  - Writing in unary might be too slow, need to make efficient
  - [Considine et al.'05]: simulate a random variable that tells which entries in sketch are set
  - [Aduri, Tirthapura '05]: allow range updates, treat (i,c_i) as range.

Other applications of FM in ODI

- Can take sketches and other summaries and make them ODI by *replacing counters with FM sketches*
  - CM sketch + FM sketch = CMFM, ODI point queries etc. [Cormode, Muthukrishnan '05]
  - Q-digest + FM sketch = ODI quantiles [Hadjieleftheriou, Byers, Kollios '05]
  - Counts and sums [Nath et al.'04, Considine et al.'05]

6	5	4	3	2	1
0	1	1	0	1	1

Combining ODI and Tree

- *Tributaries and Deltas* idea
  [Manjhi, Nath, Gibbons '05]
- Combine small synopsis of tree-based aggregation with reliability of ODI
  - Run tree synopsis at edge of network, where connectivity is limited (tributary)
  - Convert to ODI summary in dense core of network (delta)
  - Adjust crossover point adaptively

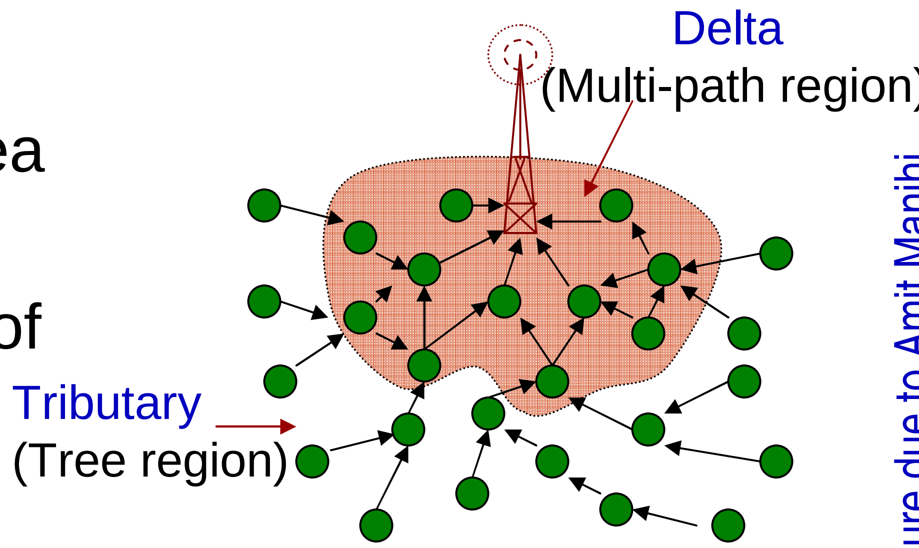
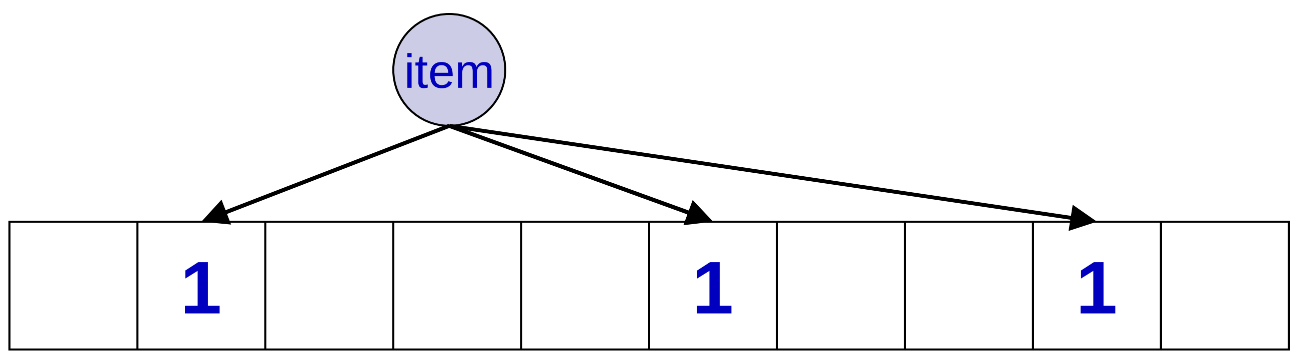


Figure due to Amit Manjhi

Bloom Filters

- Bloom filters compactly encode set membership
  - k hash functions map items to bit vector k times
  - Set all k entries to **1** to indicate item is present
  - Can lookup items, store set of size n in ~ 2n bits



- Bloom filters are ODI, and merge like FM sketches

Open Questions and Extensions

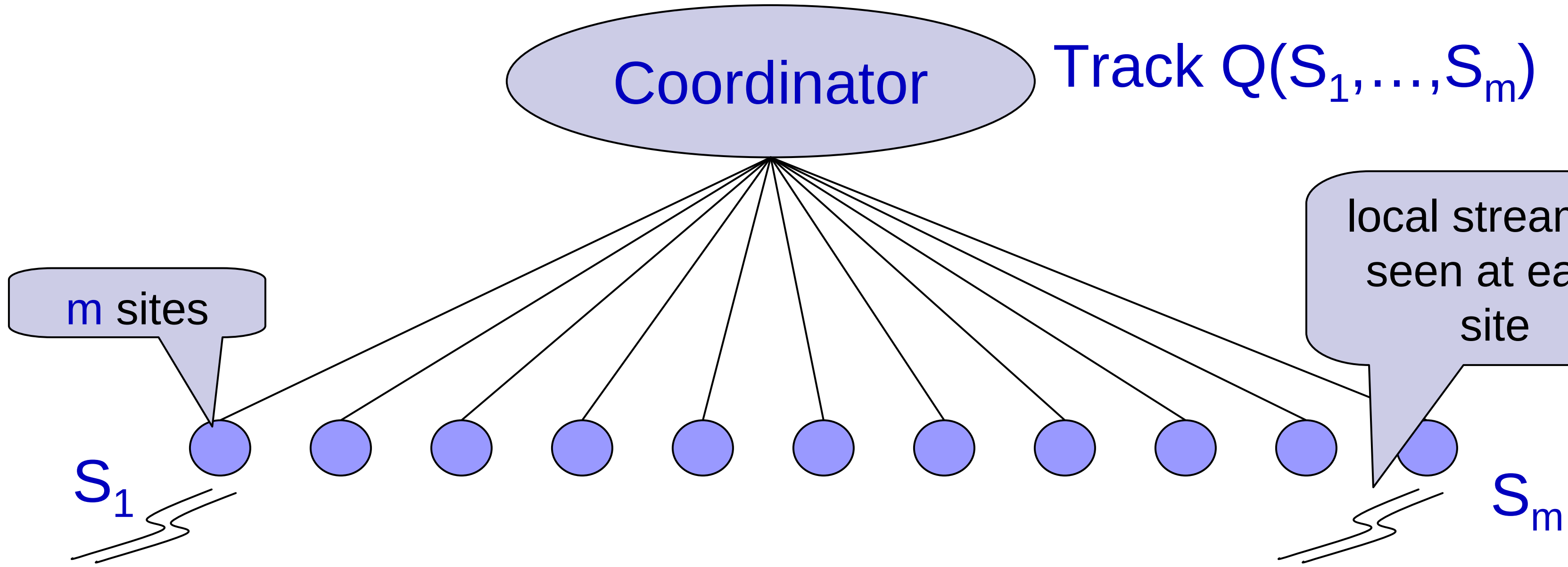
- Characterize all queries – can everything be made ODI with small summaries?
- How practical for different sensor systems?
  - Few FM sketches are very small (10s of bytes)
  - Sketch with FMs for counters grow large (100s of KBs)
  - What about the computational cost for sensors?
- Amount of randomness required, and implicit coordination needed to agree hash functions etc.?

6	5	4	3	2	1
0	1	1	0	1	1

Tutorial Outline

- Introduction, Motivation, Problem Setup
- One-Shot Distributed-Stream Querying
- Continuous Distributed-Stream Tracking
  - Adaptive Slack Allocation
  - Predictive Local-Stream Models
  - *Distributed Triggers*
- *Probabilistic Distributed Data Acquisition*
- Conclusions

Continuous Distributed Model



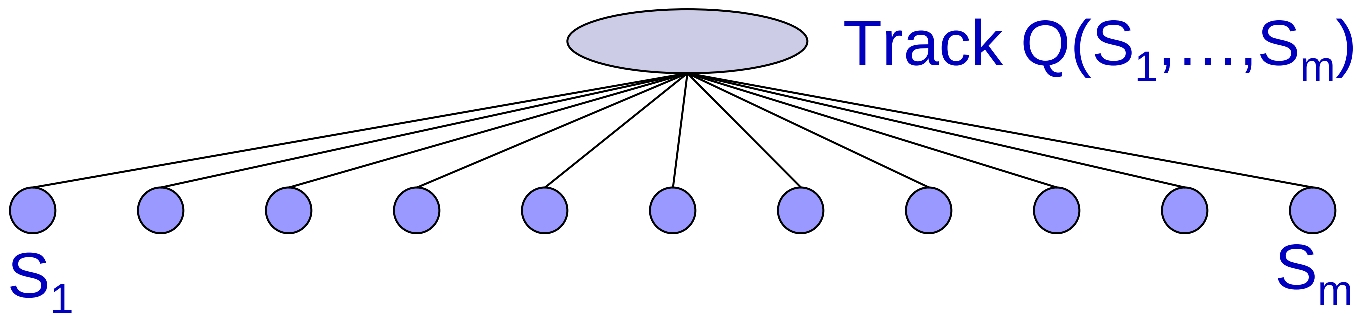
- Other structures possible (e.g., hierarchical)
- Could allow site-site communication, but mostly unneeded

Goal: *Continuously track* (global) query over streams at the coordinator

- – Large-scale network-event monitoring, real-time anomaly/ DDoS attack detection, power grid monitoring, …

Continuous Distributed Streams

- But… local site streams continuously change!
  - E.g., new readings are made, new data arrives
  - *Assumption:* Changes are somewhat smooth and gradual
- Need to guarantee an answer at the coordinator that is always correct, within some guaranteed accuracy bound
- Naïve solutions must *continuously* centralize all data
  - Enormous communication overhead!

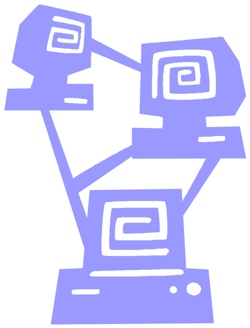


Challenges

- Monitoring is Continuous…
  - Real-time tracking, rather than one-shot query/response
- …Distributed…
  - Each remote site only observes part of the global stream(s)
  - *Communication constraints:* must minimize monitoring burden
- …Streaming…
  - Each site sees a high-speed local data stream and can be resource (CPU/memory) constrained
- …Holistic…
  - Challenge is to monitor the *complete global data distribution*
  - Simple aggregates (e.g., aggregate traffic) are easier

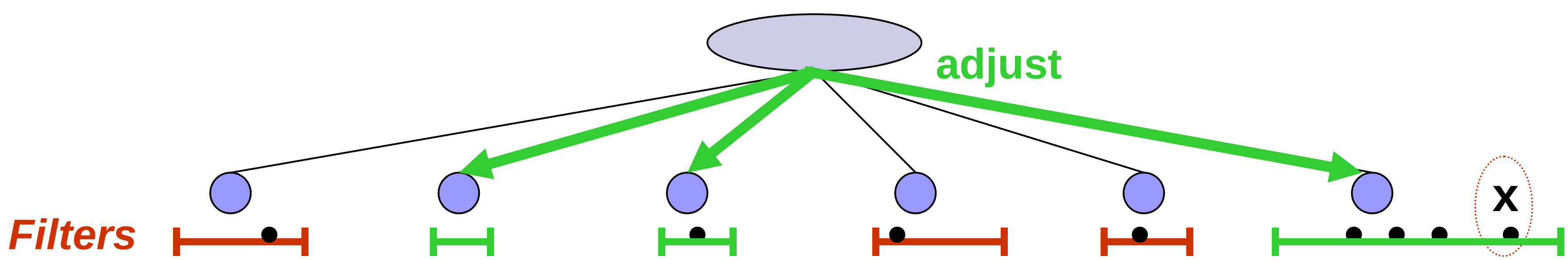
How about Periodic Polling?

- Sometimes periodic polling suffices for simple tasks
  - E.g., SNMP polls total traffic at coarse granularity
- Still need to deal with holistic nature of aggregates
- Must balance polling frequency against communication
  - Very frequent polling causes high communication, excess battery use in sensor networks
  - Infrequent polling means delays in observing events
- Need techniques to reduce communication while guaranteeing rapid response to events

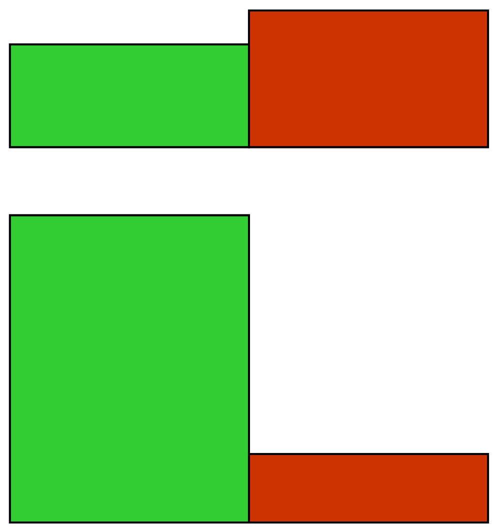


Communication-Efficient Monitoring

- Exact answers are not needed
  - Approximations with accuracy guarantees suffice
  - Tradeoff *accuracy* and *communication/ processing cost*
- Key Insight: *"Push-based" in-network processing*
  - *Local filters* installed at sites process local streaming updates
    - Offer bounds on local-stream behavior (at coordinator)
  - "Push" information to coordinator only when filter is violated
  - Coordinator sets/adjusts local filters to guarantee accuracy



Adaptive Slack Allocation



Slack Allocation

- A key idea is Slack Allocation
- Because we allow approximation, there is slack: the tolerance for error between computed answer and truth
  - May be absolute: $|Y - \hat{Y}| \leq \varepsilon$: slack is ε
  - Or relative: $\hat{Y} / Y \leq (1 \pm \varepsilon)$: slack is εY
- For a given aggregate, show that the slack can be divided between sites
- Will see different slack division heuristics

Top-k Monitoring

- Influential work on monitoring [Babcock, Olston'03]
  - Introduces some basic heuristics for dividing slack
  - Use local offset parameters so that all local distributions look like the global distribution
  - Attempt to fix local slack violations by negotiation with coordinator before a global readjustment
  - Showed that message delay does not affect correctness

Billboard Top 100

Images from http://www.billboard.com

Top-k Scenario

- Each site monitors n objects with local counts $V_{i,j}$

item $i \in [n]$

site $j \in [m]$

- Values change over time with updates seen at site j
- Global count $V_i = \sum_j V_{i,j}$
- Want to find topk, an ε-approximation to true top-k set:
  - OK provided $i \in topk$, $l \notin topk$, $V_i + \varepsilon \geq V_l$

gives a little "wiggle room"

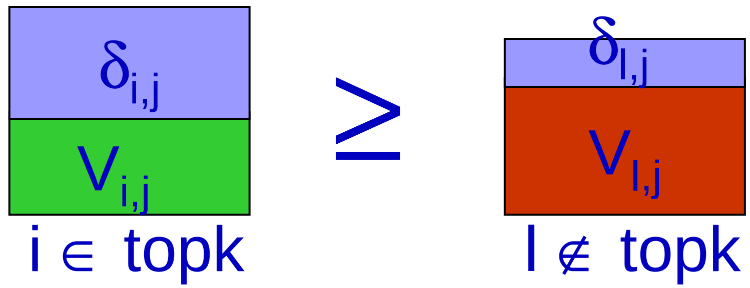
Adjustment Factors

- Define a set of 'adjustment factors', $\delta_{i,j}$
  – Make top-k of $V_{i,j} + \delta_{i,j}$ same as top-k of V_i

- Maintain invariants:
  1. For item i, adjustment factors sum to zero
  2. $\delta_{l,0}$ of non-topk item $l \leq \delta_{i,0} + \varepsilon$ of topk item i
  – Invariants and local conditions used to prove correctness

Local Conditions and Resolution

Local Conditions:
At each site j check adjusted topk counts dominate non-topk

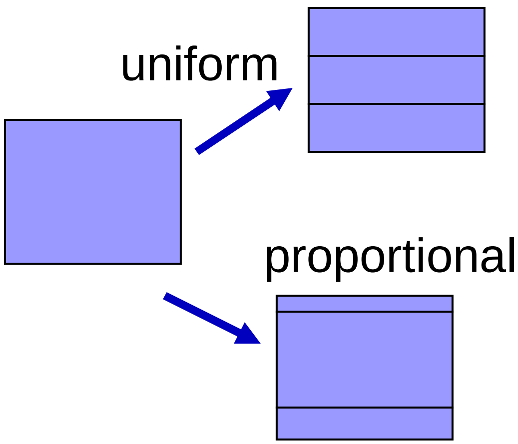


If any local condition violated at site j, resolution is triggered

- Local resolution: site j and coordinator only try to fix
  - Try to "borrow" from $\delta_{i,0}$ and $\delta_{l,0}$ to restore condition
- Global resolution: if local resolution fails, contact all sites
  - Collect all affected $V_{i,j}$s, ie. topk plus violated counts
  - Compute slacks for each count, and reallocate (next)
  - Send new adjustment factors $\delta'_{i,j}$, continue

Slack Division Strategies

- Define "slack" based on current counts and adjustments
- What fraction of slack to keep back for coordinator?
  - $\delta_{i,0} = 0$: No slack left to fix local violations
  - $\delta_{i,0} = 100\%$ of slack: Next violation will be soon
  - Empirical setting: $\delta_{i,0} = 50\%$ of slack when ε very small
    $\delta_{i,0} = 0$ when ε is large ($\varepsilon > V_i/1000$)
- How to divide remainder of slack?
  - Uniform: $1/m$ fraction to each site
  - Proportional: $V_{i,j}/V_i$ fraction to site j for i

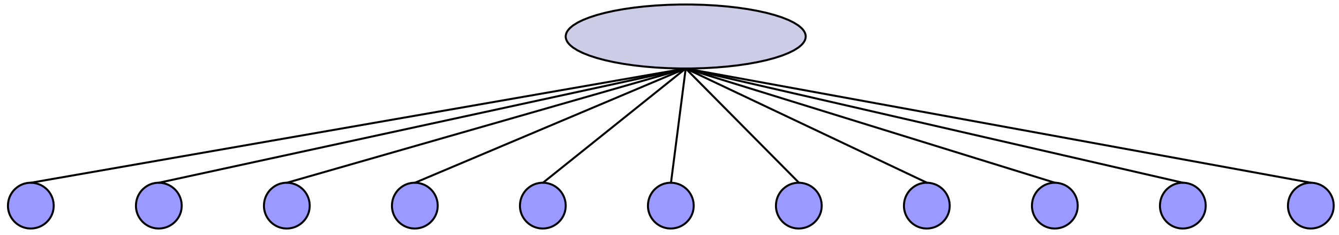


Pros and Cons

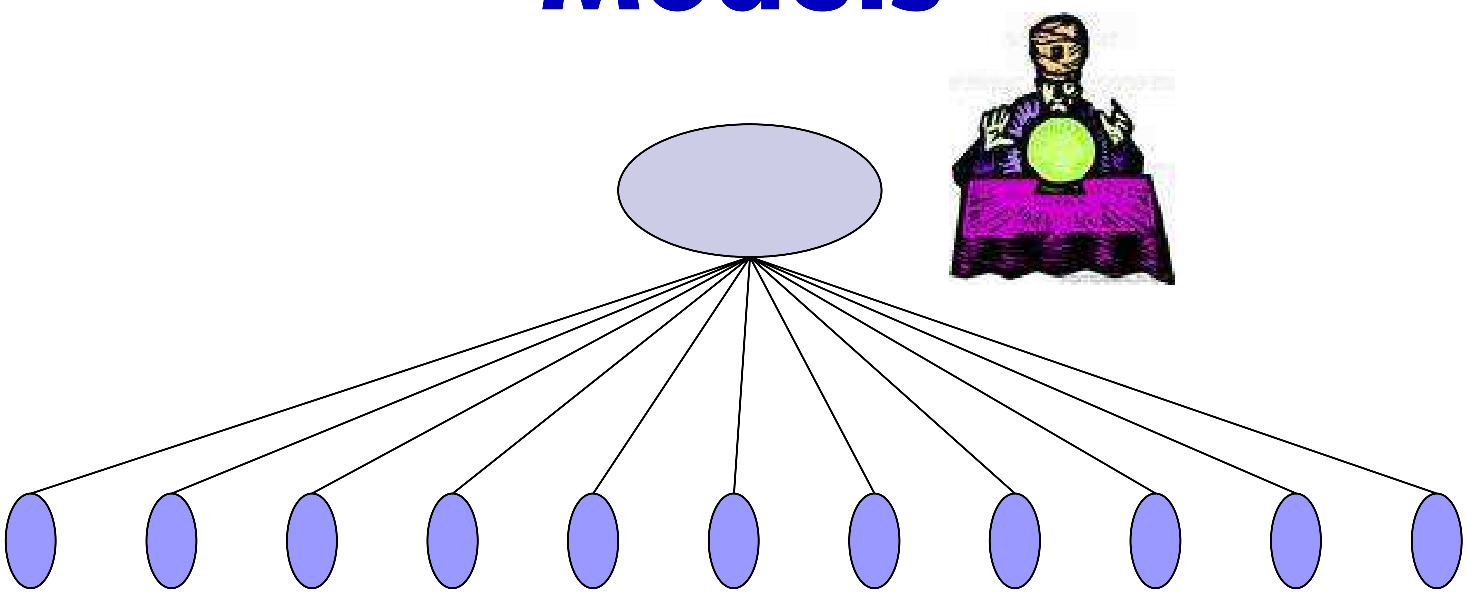
- Result has many advantages:
  - Guaranteed correctness within approximation bounds
  - Can show convergence to correct results even with delays
  - Communication reduced by 1 order magnitude (compared to sending $V_{i,j}$ whenever it changes by ε/m)
- Disadvantages:
  - Reallocation gets complex: must check $O(km)$ conditions
  - Need $O(n)$ space at each site, $O(mn)$ at coordinator
  - Large ($\approx O(k)$) messages
  - Global resyncs are expensive: m messages to k sites

General Lessons

- Break a global (holistic) aggregate into *"safe"* local conditions, so local conditions ⇒ global correctness
- Set local parameters to help the tracking
- Use the approximation to define slack, divide slack between sites (and the coordinator)
- Avoid global reconciliation as much as possible, try to patch things up locally

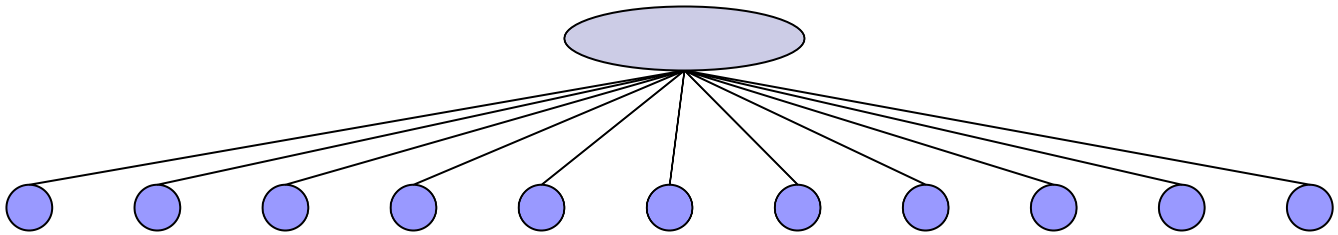


Predictive Local-Stream Models

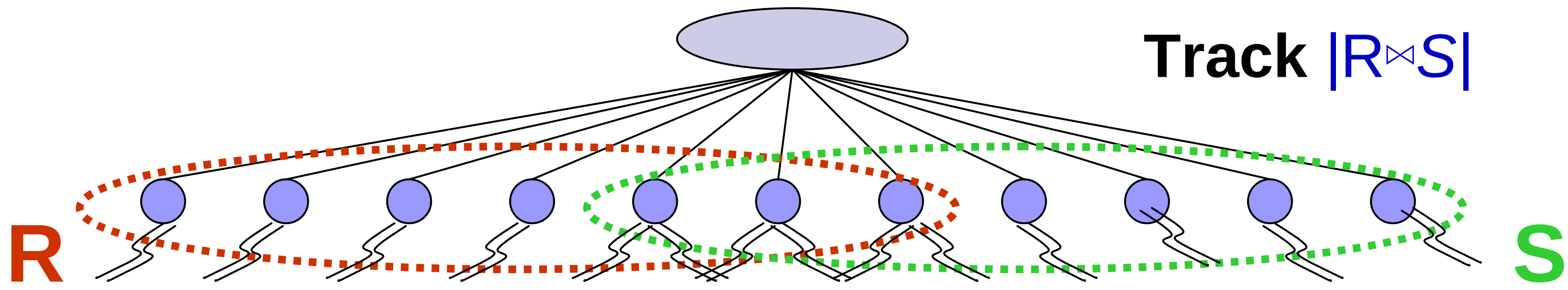


More Sophisticated Local Predictors

- Slack allocation methods use simple *"static"* prediction
  - Site value implicitly assumed constant since last update
  - No update from site ⇒ last update ("predicted" value) is within required slack bounds ⇒ global error bound
- *Dynamic, more sophisticated prediction models* for local site behavior?
  - Model complex stream patterns, reduce number of updates to coordinator
  - But... more complex to maintain and communicate (to coordinator)



Tracking Complex Aggregate Queries



- Continuous distributed tracking of complex aggregate queries using AMS sketches and local prediction models [Cormode, Garofalakis'05]
- *Class of queries:* Generalized inner products of streams

$$|R \bowtie S| = f_R \cdot f_S = \sum_v f_R[v]\, f_S[v] \qquad (\pm\, \varepsilon\, \|f_R\|_2\, \|f_S\|_2)$$

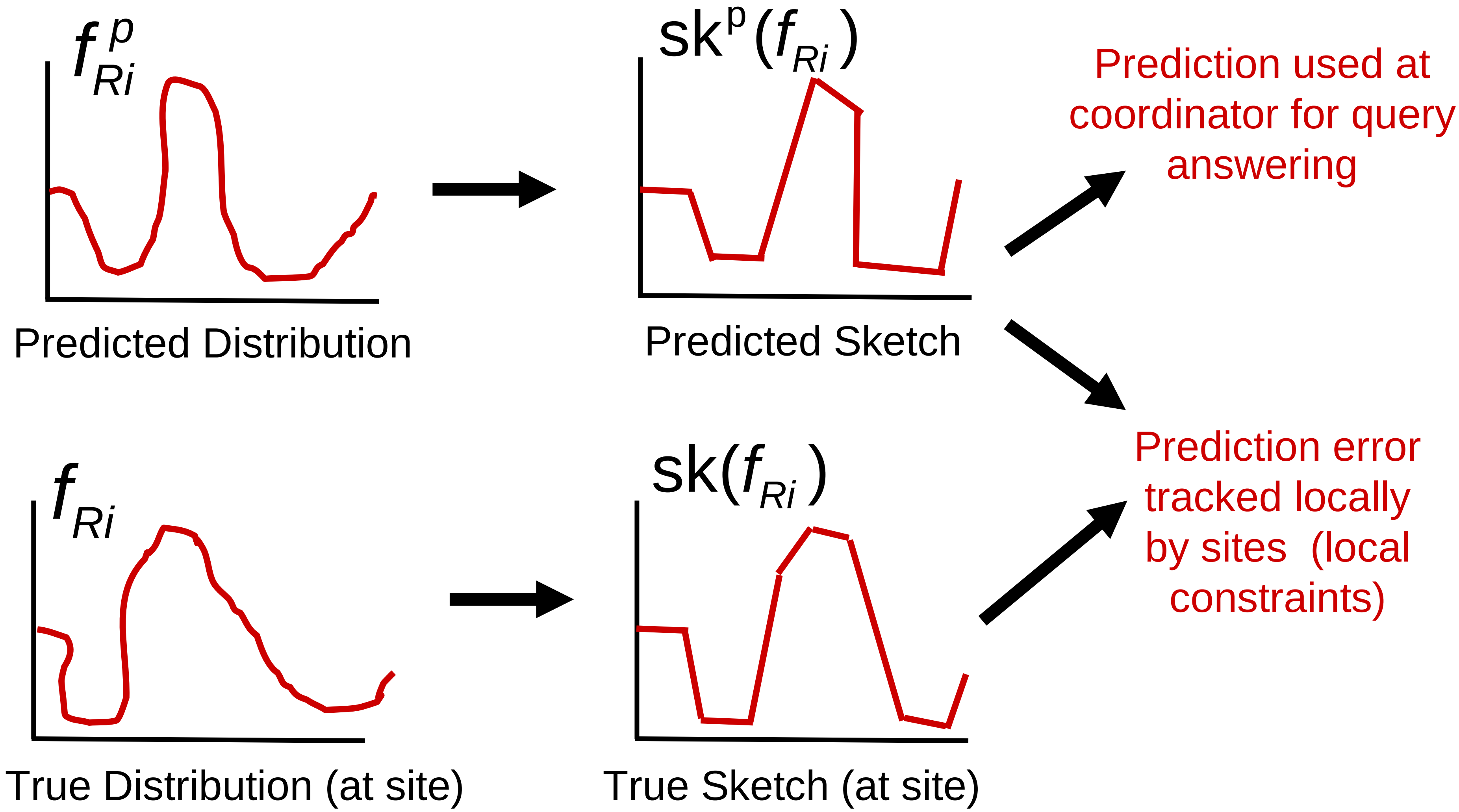
  - Join/multi-join aggregates, range queries, heavy hitters, histograms, wavelets, …

Local Sketches and Sketch Prediction

- Use (AMS) sketches to summarize local site distributions
  - Synopsis=small collection of random linear projections $sk(f_{R,i})$
  - *Linear transform:* Simply add to get global stream sketch

- Minimize updates to coordinator through *Sketch Prediction*
  - Try to predict how local-stream distributions (and their sketches) will evolve over time
  - Concise *sketch-prediction models*, built locally at remote sites and communicated to coordinator
  - *Shared knowledge* on expected stream behavior over time: Achieve "stability"

Sketch Prediction



Query Tracking Scheme

Tracking. At site j keep sketch of stream so far, $sk(f_{R,i})$

- Track local deviation between stream and prediction:

$$\| sk(f_{R,i}) - sk^p(f_{R,i}) \|_2 \leq \theta/\sqrt{k} \, \| sk(f_{R,i}) \|_2$$

- Send current sketch (and other info) if violated

Querying. At coordinator, query error $\leq (\varepsilon + 2\theta)\|f_R\|_2 \, \|f_S\|_2$

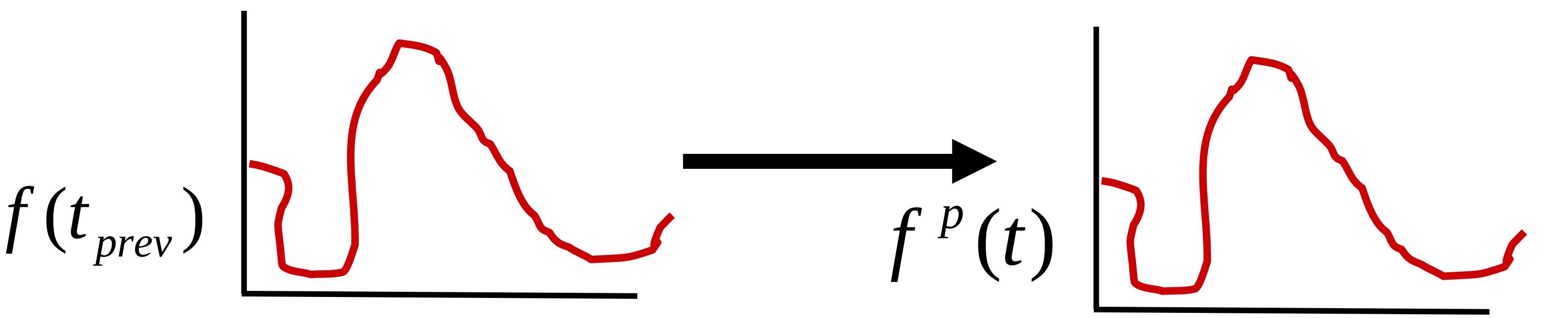
- ε = local-sketch summarization error (at remote sites)
- θ = upper bound on local-stream deviation from prediction ("Lag" between remote-site and coordinator view)

- ***Key Insight:*** *With local deviations bounded, the predicted sketches at coordinator are guaranteed accurate*

Sketch-Prediction Models

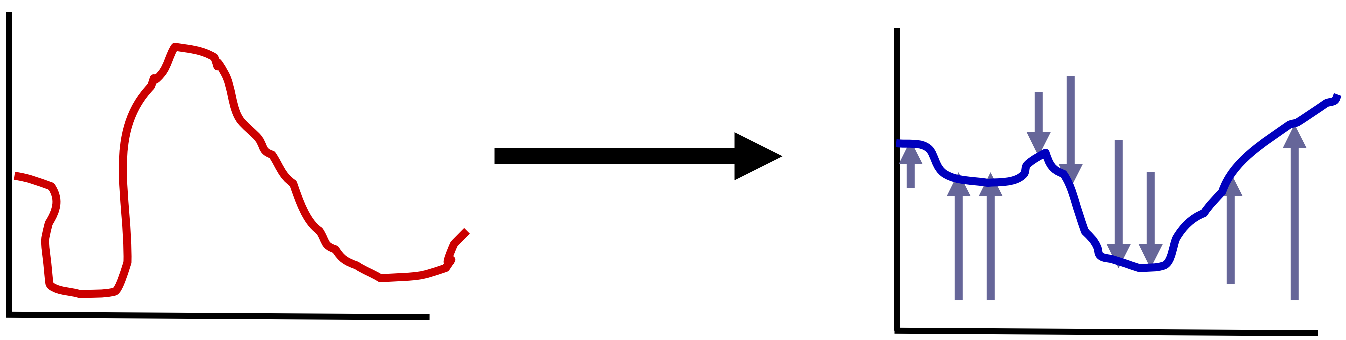
- Simple, concise models of local-stream behavior
  - Sent to coordinator to keep site/coordinator "in-sync"
  - Many possible alternatives

- Static model: No change in distribution since last update
  - Naïve, "no change" assumption:
  - No model info sent to coordinator, $sk^p(f(t)) = sk(f(t_{prev}))$



Sketch-Prediction Models

- Velocity model: Predict change through “velocity” vectors from recent local history (simple linear model)
  - Velocity model: $f^p(t) = f(t_{prev}) + \Delta t \bullet v$
  - By sketch linearity, $sk^p(f(t)) = sk(f(t_{prev})) + \Delta t \bullet sk(v)$
  - Just need to communicate one extra sketch
  - Can extend with acceleration component



$$f(t_{prev}) \qquad\qquad f^p(t) = f(t_{prev}) + \Delta t \cdot v$$

Sketch-Prediction Models

Model	Info	Predicted Sketch
Static	∅	$\mathrm{sk}^{p}(f(t)) = \mathrm{sk}(f(t_{prev}))$
Velocity	sk(v)	$\mathrm{sk}^{p}(f(t)) = \mathrm{sk}(f(t_{prev})) + \Delta t \cdot \mathrm{sk}(v)$

- 1 – 2 orders of magnitude savings over sending all data

Lessons, Thoughts, and Extensions

- Dynamic prediction models are a natural choice for continuous in-network processing
  - Can capture complex temporal (and spatial) patterns to reduce communication
- Many model choices possible
  - Need to carefully balance power & conciseness
  - Principled way for model selection?
- General-purpose solution (generality of AMS sketch)
  - Better solutions for special queries
    E.g., continuous quantiles [Cormode et al.'05]

Conclusions

- Many new problems posed by developing technologies
- Common features of *distributed streams* allow for general techniques/principles instead of “point” solutions
  - In-network query processing
    Local filtering at sites, trading-off approximation with processing/network costs, …
  - Models of “normal” operation
    Static, dynamic (“predictive”), probabilistic, …
  - Exploiting network locality and avoiding global resyncs
- Many new directions unstudied, more will emerge as new technologies arise
- *Lots of exciting research to be done*! ☺