Feather : Hierarchical Querying for the edge

Seyed Hossein, Mohammad Salehe

Moshe Gabel, Eyal de Lara

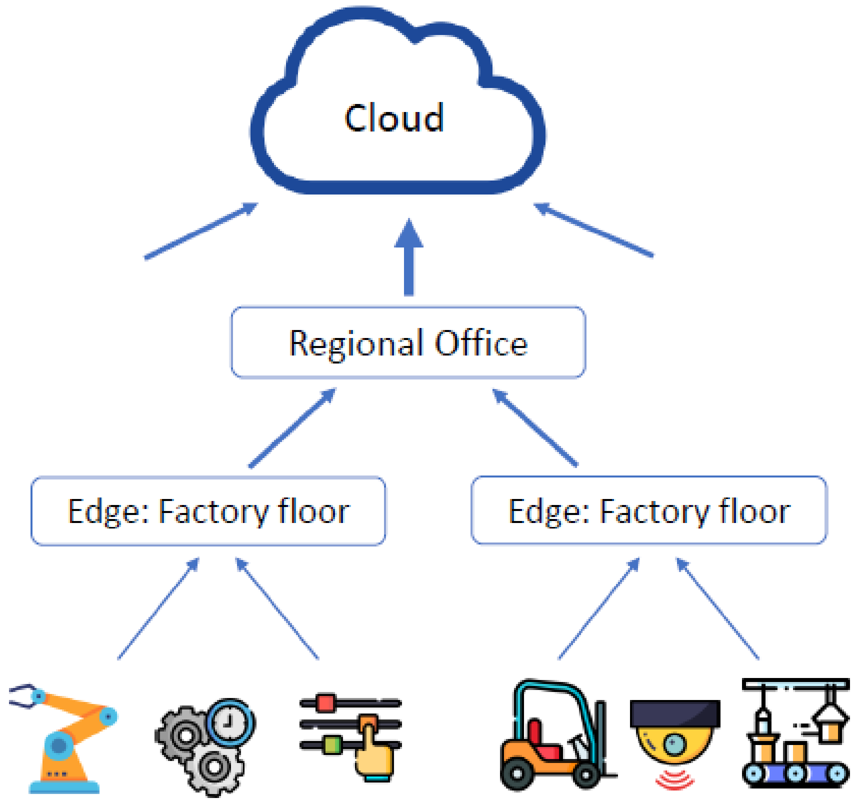
University Of Toronto

Data on the Edge

- Data is generated over a wide geographic area
  - Is stored near the edges
  - Pushed periodically upstream to a hierarchy of data centres

- Network properties:
  - Limited bandwidth
  - High latency
  - Failures

- Observation: Queries in general are less latency sensitive as you move away from the edge

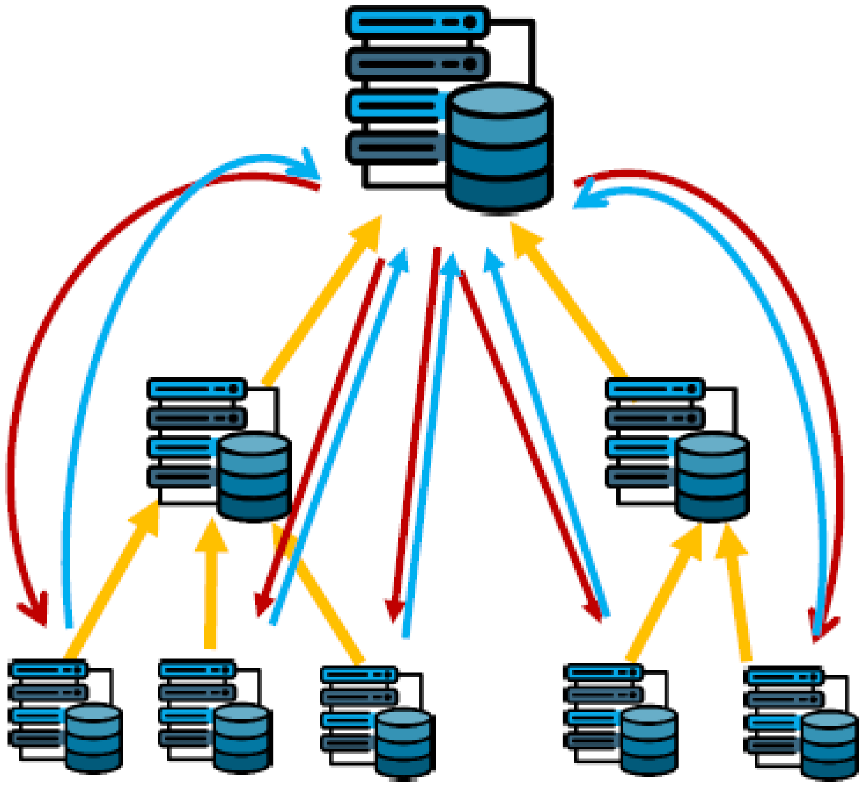


Feather - Overview

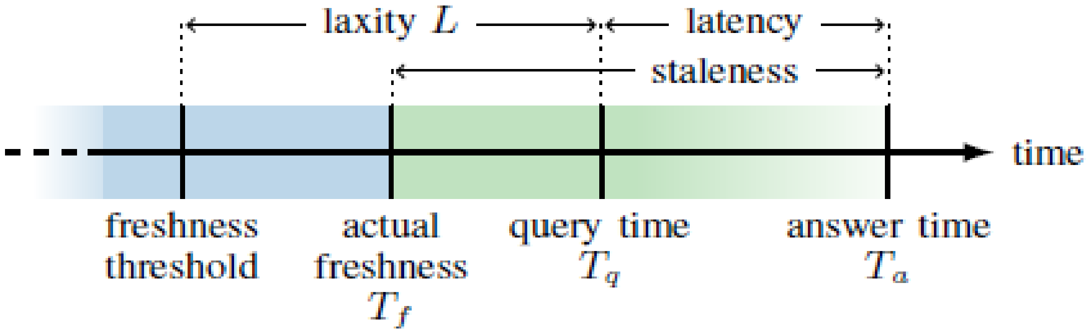
- Allows users to intelligently control the trade-off between data freshness and query answer latency.

- Users can specify precise freshness constraint for each individual query, or alternatively a deadline for the answer.

- Applications : Urban sensing, Smart grid, Industrial automation etc.



Terminology

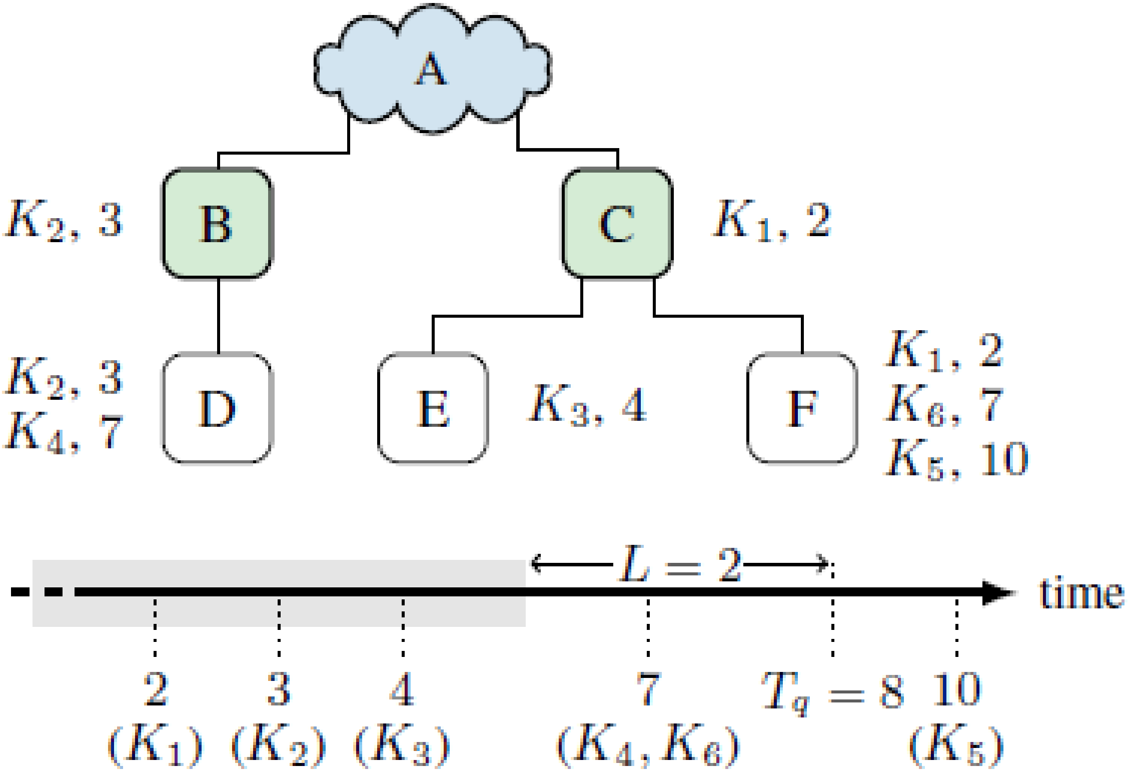



```
SELECT AVG(power) FROM hardwareStats
WHERE machine = 'arm robot'
      AND timestamp >= NOW()-600
LAXITY = 30
```

- Local v/s Global Queries :
  - Local queries are fast reads and writes executed directly on the high-performance local data store
  - Global queries are on-demand read queries that provide user-specified freshness guarantees

- User provides minimum freshness requirement - Laxity
- System guarantees answer is at least as fresh – Staleness = Ta – Tf
- Latency = Ta - Tq

Example – Trade-off b/w Laxity and Latency



Answering Global Queries

Algorithm 1: The hierarchical algorithm for global queries with freshness guarantee L.

Input: query q, query time T_q, laxity L, current node n
Output: result R, actual freshness time T_f

Initialize set of accessed children $A \leftarrow \emptyset$
Initialize result R
foreach *child* $c \in \text{children}(n)$ **do**
    if *last update time from child* $T_u(c) < T_q - L$ **then**
        Add c to accessed children: $A \leftarrow A \cup \{c\}$
        Send global query q to child c
$R_{loc} \leftarrow$ execute q on local store on rows not from A
Update result R with local results R_{loc}
Set freshness time T_f to latest update time:
 $T_f \leftarrow \min_c\{T_u(c)\}$
foreach *response* R_c, T_c *from child* c *of node* n **do**
    Update result R with child result R_c
    $T_f \leftarrow \min(T_f, T_c)$
Return results R and actual freshness time T_f

Providing Latency guarantees

- Latency guarantee is achieved by treating nodes that did not respond in time as failed links.
- Modification to the Algorithm :
  - When a node receives query with a deadline, it decreases the deadline and sends this to the child to make some headroom for processing delays etc.
  - Additionally it queries the child data present on local store. This result is used when the child does not return result within the deadline.

```
SELECT AVG(power) FROM hardwareStats
WHERE machine = 'arm robot'
      AND timestamp >= NOW()-600
~~LAXITY = 30~~  DEADLINE = 150
```

Result Set Coverage

- Feather provides analytical information on
  - how many nodes participated in the querying process,
  - how many data rows were included in the query
  - an estimate of the number of updated data rows that were not included in the query due to freshness constraints or link errors.

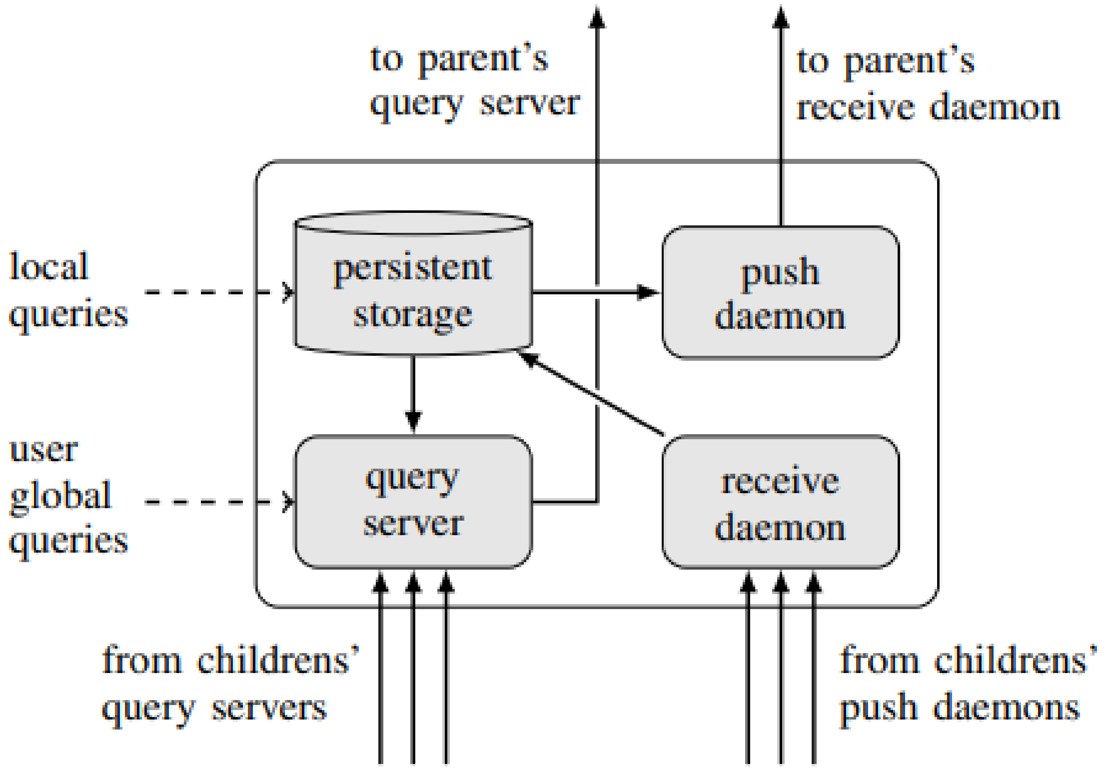
$$\rho(c) = \frac{\sum_{i=0}^{K-1} R_i(c)}{T_0(c) - T_K(c)}$$

$$\rho(c) \cdot (t - T_0(c)).$$

Handling Failures

- If a link to a child that must be queried has failed or a sub-query timed-out, then we cannot provide the freshness guarantee for that particular query.
- Feather provides either:
  - A complete but less fresh answer that includes old results for the missing child. ($T_f < T_q - L$)
  - Or a partial but up-to-date answer. ($T_f > T_q - L$)

Architecture



Writes & Replication

- User applications write data directly to the Feather local store at the node they are running at.
- To support replication and querying, the following columns are added to the client applications’ schema, and added to user writes by a client side driver:
  - a timestamp column;
  - a Boolean dirty
  - a prev_loc that determines from which node the row was received from.
- Push daemon reads dirty rows and pushes up the hierarchy.

Global Queries

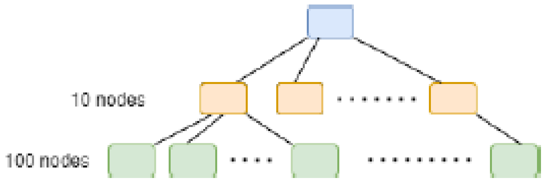
- Uses Cassandra for persistent local storage.
- Supports almost all features provided by CQL, specifically all aggregate functions (*, MAX, MIN, AVG, COUNT, SUM) and most clauses (WHERE, GROUP BY, ORDER, LIMIT, DISTINCT)
- IN Clause – Materialised view.
- Can query on data from specific children.

```
SELECT * FROM table WHERE key = value
  AND timestamp > NOW() - L
  AND prev_loc IN ('F','C')
```

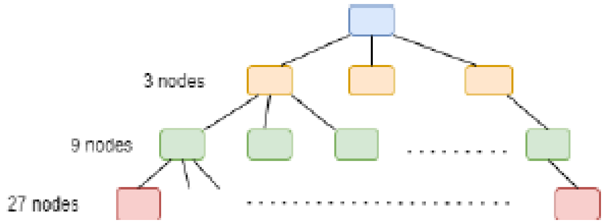
Evaluation

- Metrics:
  - Latency : Ta−Tq.
  - Staleness : Ta − Tf .
  - Bandwidth: total number of rows sent over all links in the edge network.
  - Work at edges: average number of rows retrieved from edge nodes
  - Coverage estimation accuracy: estimate how many data rows were needed to answer the query

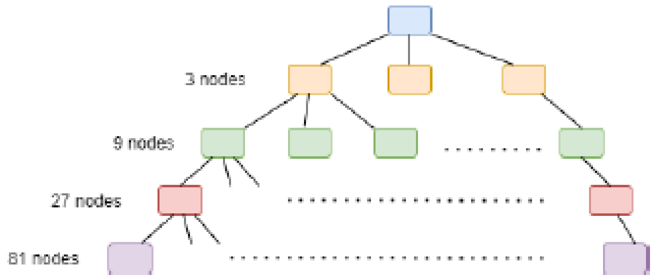
Experimental Setup



Wide topology



Medium topology

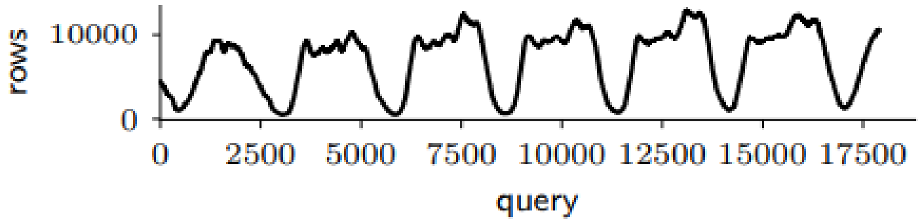


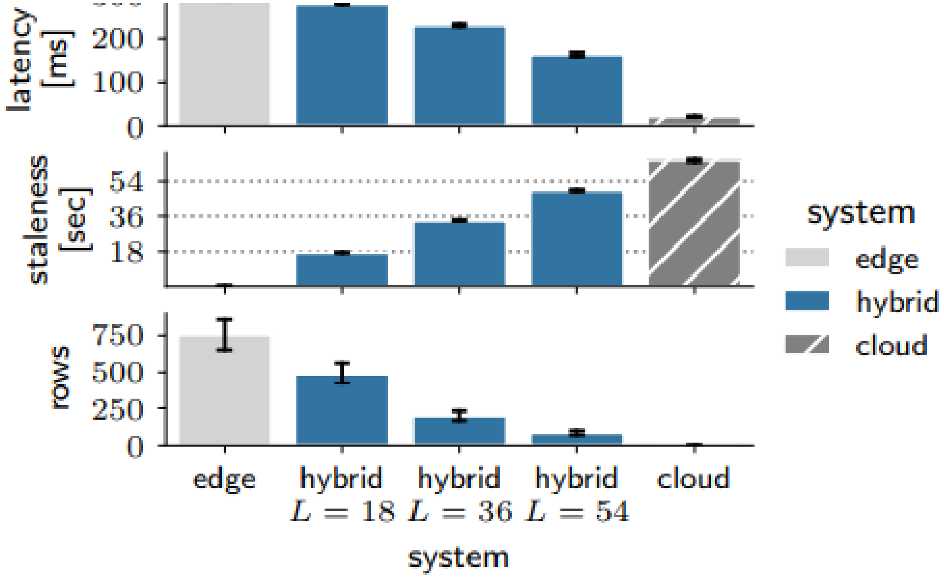
Deep topology

- New York Taxi Dataset - 7 million rides. Contains geo-distributed labelled data (pick-up and drop-off zones), as well as information such as fare amount etc.
- When inserting data rows, row's drop-off zone is used to determine which edge node to add it to. The dataset contains 265 such geographical zones.
- Issue 3 queries(SELECT, MIN, GROUPBY) on the data, all filtered to a window of the last 90 seconds of real time. (45 min real time)

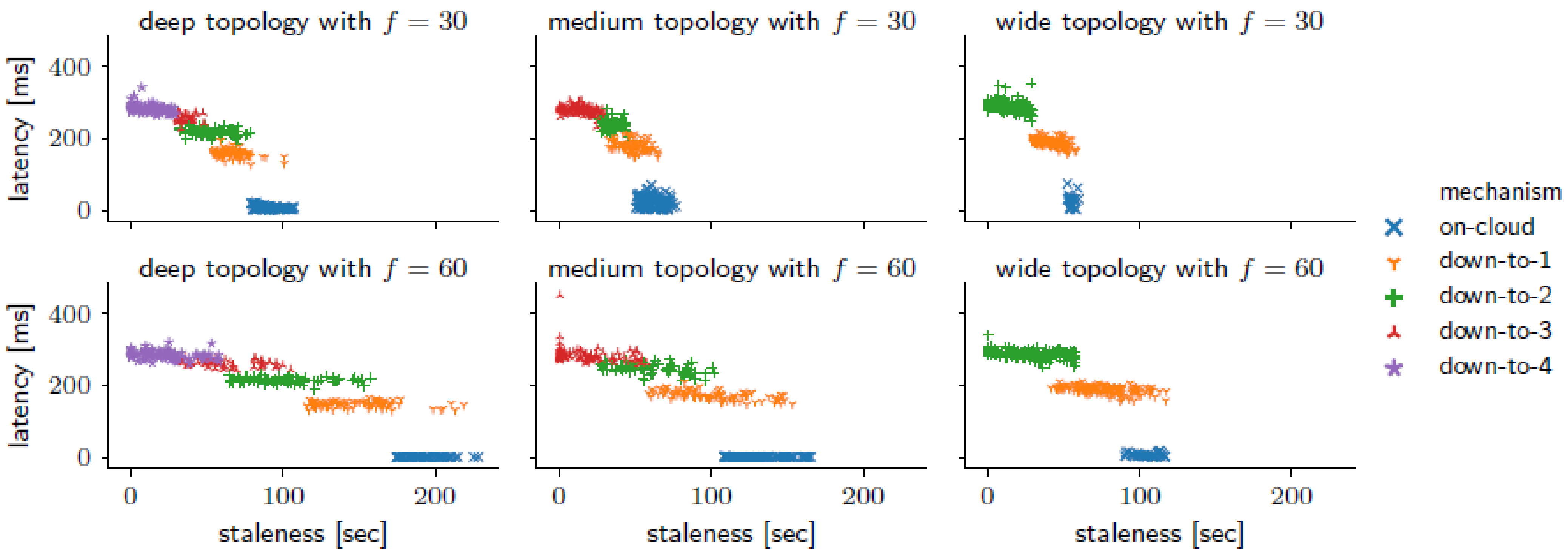
Results

- Feather is run for 18000 seconds approx. 1 week real time.
- Fig 1 shows the number of rows covered by the 90 second window in each such query
- Every second, one query is issued with laxity set between 0 and (D − 1) · f where D is the depth of the topology and f is the period of the push demon.



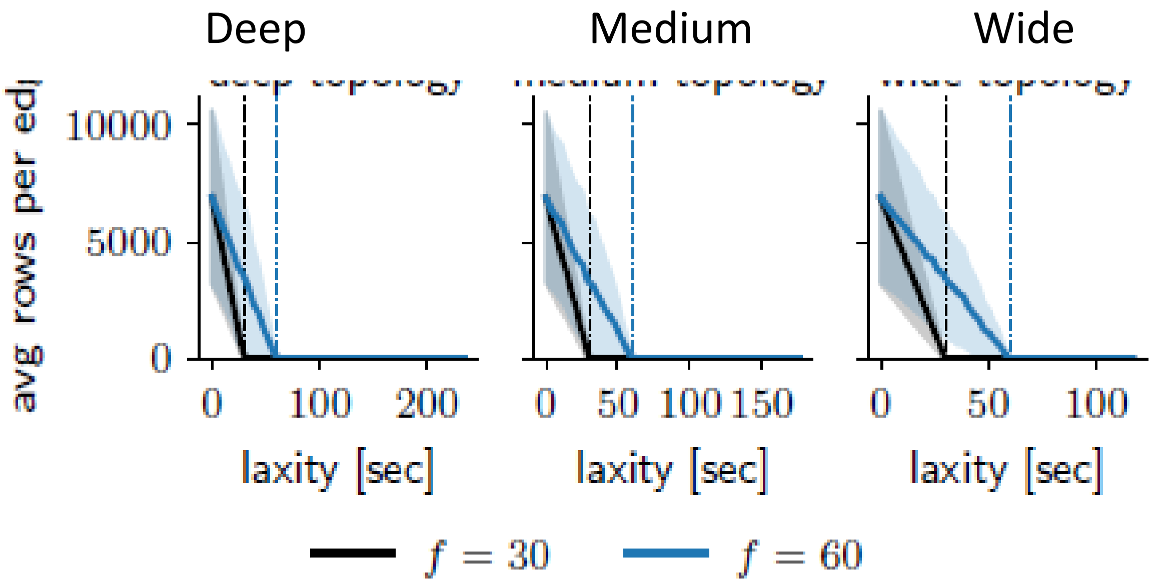


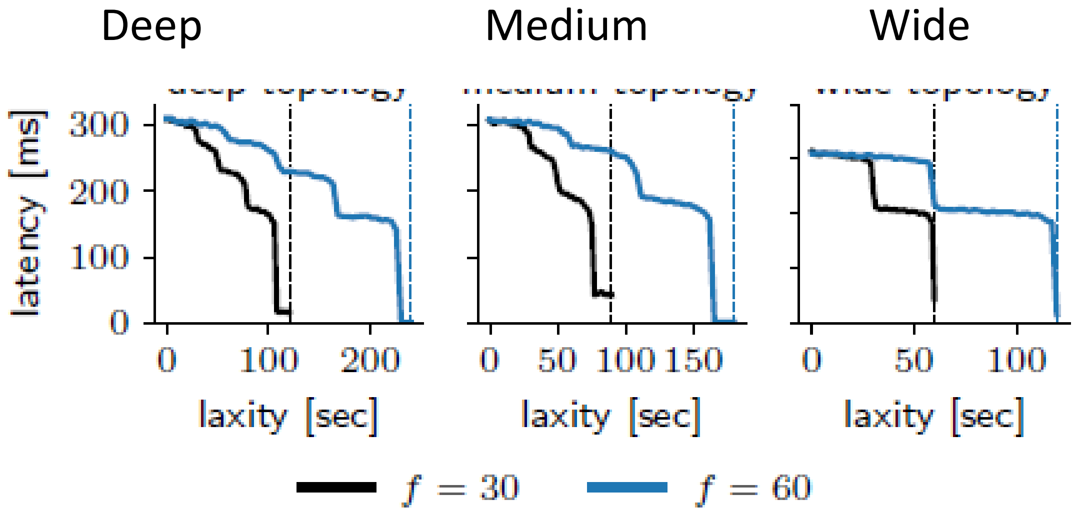
Results



Trade-off b/w latency and staleness depends on query laxity, network topology, period of the push demon and data update distribution among the edges.

Results


Deep
Medium
Wide
avg rows per ed
10000
5000
0
0
100
200
0
50
100
150
0
50
100
laxity [sec]
laxity [sec]
laxity [sec]
$f = 30$
$f = 60$


Deep
Medium
Wide
latency [ms]
300
200
100
0
0
100
200
0
50
100
150
0
50
100
laxity [sec]
laxity [sec]
laxity [sec]
$f = 30$
$f = 60$

Real world Experiment

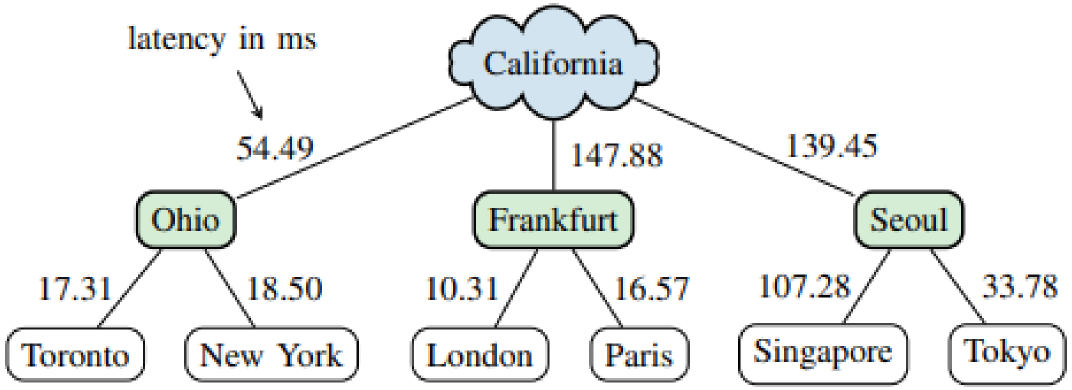
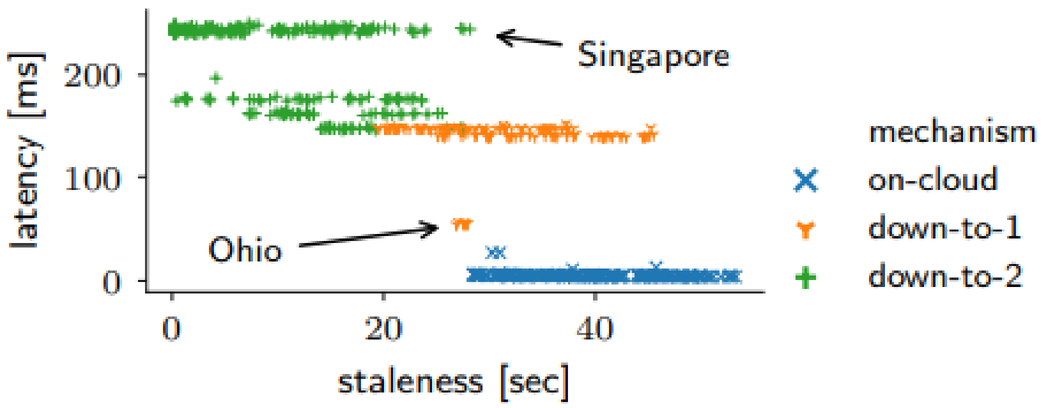


Fig. 13. Topology of the real-world experiment. Numbers indicate mean measured round trip time in milliseconds.



- geo-tagged public tweets is used as dataset
- Run over 33000 queries at a rate of 1 query per second, and set the push daemon period to f = 30 seconds.

Discussion

- Are Adhoc queries more frequent than repetitive queries?
- Is disjoint data assumption valid?
- How to handle deletions?
- Downstream data?