Scalable Ranked Publish/Subscribe

Ashwin Machanavajjhala,
Erik Vee
Minos Garofalakis,
Jayavel Shanmugasundaram
• Many subscribers, each specify some target of interest
  – E.g. Company looking for nursing employees, where job pays $40-$60/hr and work is 20-30 hrs/week
• Events arrive, each labeled with a number of attributes
  – E.g. Job seeker, looking for a nursing job paying $50/hr and 25 hours/week
• Subscribers notified about every event they target
  – E.g. All matching companies notified about job seeker
Traditional Pub/Sub (Geometric view)

Subscribers specify rectangle in high-dimensional space

May preprocess rectangles
Subscribers specify rectangle in high-dimensional space

Event describes a point in the space

Return every rectangle “stabbed” by the point
The same example on the web

- Companies are looking for potential employees
  - Specify some target attributes
- Users arrive, looking for jobs
  - Specify some attributes
- User is shown companies that match his search
- BUT—*only top 5* are shown due to space limitations
- Same space limitations for applications like display advertising, load shedding
Ranked Pub/Sub Problem

• Given a set of subscriptions:
  – Each subscription describes a rectangle in high-dim space
    • Each attribute corresponds to a dimension
  – Each subscription gets a score
    • May be static, or function of attribute scores
  – Allowed to preprocess

• Events arrive online:
  – Each event describes a point in high-dim space
  – Each event also associated with a value k

• Return the k highest-scoring subscribers
Our focus

- Examine range queries in single dimensional case
  - Subscribers specify intervals (and score)
  - Events are 1-dim points
- Single dimension is building block for multi-dimensional case
  - If score is static across attributes, do standard list intersection
  - If score function of attribute-scores, apply threshold algorithm
Our focus

- Examine range queries in single dimensional case
  - Subscribers specify intervals (and score)
  - Events are 1-dim points
- Single dimension is building block for multi-dimensional case

- Restrict our attention to small memory structures
  - i.e. Intervals never broken into pieces (hence, linear space)
- Propose several novel data structures
- Compare these structures with variants of standards
  - Show marked improvement for low dimensional problems
  - Do well even compared to larger-memory structures
Standard structures for pub/sub

- Interval Tree
- R-Tree
- Segment Tree
  - Space blow-up is $O(\log n)$
  - This is actually an issue—our experiments showed an order of magnitude larger memory footprint
Reminder...

Interval Trees

Intervals
higher = higher score
Reminder…

*Interval Trees*

Intervals higher = higher score

Pick a stabbing line
Pick a stabbing line

All stabbed intervals go into one node

Left intervals

Right intervals
Reminder…

**Interval Trees**

Intervals higher = higher score

Pick a stabbing line

All stabbed intervals go into one node

Repeat on left and right intervals
Reminder…

Interval Trees

Intervals higher = higher score

Stabbing points

Pick a stabbing line

All stabbed intervals go into one node

Repeat on left and right intervals

For each node, store intervals sorted by left endpoint and sorted by right endpoint
R-Trees

Intervals
higher = higher score
Score sorted R-Trees

Each node stores containing interval

branching factor = 3

Group intervals by score

Intervals higher = higher score

[30,100]

[30,100]
Score sorted R-Trees

Each node stores containing interval

branching factor = 3

Group intervals by score

Intervals higher = higher score
Score sorted R-Trees

- Each node stores containing interval
- **branching factor = 3**
- Group intervals by score
- Intervals higher = higher score
Score sorted R-Trees

Each node stores containing interval

branching factor = 3

Group intervals by score

Intervals higher = higher score

For a query, output first k hits
Score sorted R-Trees

Each node stores containing interval

branching factor = 3

Group intervals by score

Intervals higher = higher score

May have many “holes” = wasted probes
Segment trees

All intervals broken into segments, based on set of endpoints
Segment trees

All intervals broken into segments, based on set of endpoints

Form tree on segments

Each node records segment: \([a, b]\)
Segment trees

Form tree on segments

Each node records segment: \([a, b]\)

Advantage: if interval stored at node, then interval contains all of \([a, b]\)

So each node stores intervals in \textit{score-sorted} order!

All intervals broken into segments, based on set of endpoints
Standard structures for ranked pub/sub

• Interval Tree
  – Sort intervals by score, or by interval— not both

• R-Tree
  – Scored R-tree
  – “Holes” can get you

• Segment Tree
  – Space blow-up is $O(\log n)$
    • This is actually an issue— our experiments showed an order of magnitude larger memory footprint
  – “Gold standard”: Scoring is no problem!
Our data structures

- IR-tree
  - Interval tree with R-tree sitting in each node

- OptR-tree
  - R-tree, but with intervals sorted to support scoring in an optimized way

- Main insight—R-trees in 1 dimension very fast, except for the wasted probes (i.e. “holes”)
  - Both data structures use R-trees, with guarantees on number of wasted probes
IR-tree

- Form basic tree as an interval tree
- For each node, index the intervals with an R-tree
• Why index by R-trees?
• Key lemma: All intervals at a node overlap, so the R-tree has no holes! (i.e. Every probe in the R-tree leads to a valid interval)

• R-trees also lightweight, simple, good in practice

• Each \texttt{getNext()} call takes at most \(O(\log \log n + \text{height}(\text{R-tree}))\)
Opt-R-Trees

- Data structure is a R-tree
- However, we can sort the intervals more intelligently
- Key insight: If two intervals do not overlap, then can interchange order
Opt-R-Trees

• Intervals induce a topological graph
  – (Edge from i1 to i2 if \( \text{score}(i1) > \text{score}(i2) \) AND \( i1, i2 \) overlap
  – We give a way of constructing taking time \( O(n \log n) \) by ignoring some transitive edges

• Any grouping that respects this graph is okay
  – We take left-most interval with indegree 0 at each step

• Key lemma: To get top k intervals, need at most 2k probes
  – Roughly, there is a hole only when there must be one
Experiments

• Used synthetic data
• 1M intervals
• Left endpoint and length of interval zipfian distributed
  – Vary the skew, zipfian power
• Looked at varying number of dimensions
Speed vs. overlap (Threshold algorithm in 4 dimensions)

Lots of overlap

Little overlap

More holes (R-tree)

Score matters less (Interval tree)

Time Taken per query (ms)

skew-length

Interval
R
S
IR
OptR
Time for getNext()
Dimensionality (Threshold algorithm)

At larger number of dimensions, all methods have similar time

More overhead, more getNext() calls
Experimental summary

• IR-trees, OptR-trees, and segment trees are all comparable in speed
  – Segment trees require too much memory
  – Only IR-trees are easy to update intervals online

• Standard structures much slower in general
Conclusions

• Propose a new problem: Ranked Pub/Sub
• Give a novel solution for one dimension
  – Yields solutions for small dimensionality
• Data structure are lightweight, easy to implement, give good results
  – IR-trees: easy to maintain

• Open problems:
  – How do we extend this to larger dimensionality?
  – More expressive subscriptions, events
  – Score updates