Approximate Range Thresholding

Zhuo Zhang†, Junhao Gan†, Zhifeng Bao‡, Seyed Mohammad Hussein Kazemi†, Guangyong Chen⋆, Fengyuan Zhu∗

†The University of Melbourne
‡RMIT University
⋆Zhejiang Lab & Zhejiang University
∗Kaifeng Investment

June, 2022

An example: Stock Trading System Scenario

Notify me when in total 1000 *shares (from now) of (APPL:NSQ) are sold at prices in* $[\$140, \$150)$.

An example: Stock Trading System Scenario

Notify me when in total 1000 *shares (from now) of (APPL:NSQ) are sold that satisfy:*

- *the selling price is in* $[\$140, \$150)$.
- *when transaction happens the price of (GOOG:NSQ) is in* $[\$60, \$70)$

Such queries can be formalized as the **Range Thresholding (RT) Problem**.

Problem Definition: Range Thresholding Problem

- **Element** e:
  - a *value* $v(e)$, a d-dimensional point, *e.g. the selling price*;
  - a *weight* $w(e)$, a positive integer, *e.g. number of shares*.

Problem Definition: Range Thresholding Problem

- **Element** e:
  - a *value* $v(e)$, a d-dimensional point, *e.g. the selling price*;
  - a *weight* $w(e)$, a positive integer, *e.g. number of shares*.
- **Stream** S:
  - a sequence of elements $e_1, e_2, \cdots$
  - e_i arrives at time stamp i

Problem Definition: Range Thresholding Problem

- **Element** e:
  - a *value* $v(e)$, a d-dimensional point, *e.g. the selling price*;
  - a *weight* $w(e)$, a positive integer, *e.g. number of shares*.
- **Stream** S:
  - a sequence of elements $e_1, e_2, \cdots$
  - e_i arrives at time stamp i
- **Query** q:
  - a *range* $R(q)$, a d-dimensional axis-parallel rectangular range, *e.g. sensitive price interval*;
  - a *threshold* $\tau(q)$, a positive integer, *e.g. the total number of shares*.

Problem Definition: Range Thresholding Problem

- The **maturity moment** of q:
  - The first time stamp that $\sum_{\substack{\text{e arrives after q} \\ v(e) \in R(q)}} w(e) \geq \tau(q)$

Problem Definition: Range Thresholding Problem

- The **maturity moment** of q:
  - The first time stamp that $\sum_{\substack{\text{e arrives after q} \\ v(e)\in R(q)}} w(e) \geq \tau(q)$
- The ε-**maturity period** of q is a time period between
  - The first time stamp that $\sum_{\substack{\text{e arrives after q} \\ v(e)\in R(q)}} w(e) \geq (1-\varepsilon)\tau(q)$
  - The first time stamp that $\sum_{\substack{\text{e arrives after q} \\ v(e)\in R(q)}} w(e) \geq \tau(q)$

Problem Definition: Range Thresholding Problem

- The **maturity moment** of q:
  - The first time stamp that $\sum_{\substack{\text{e arrives after q} \\ v(e) \in R(q)}} w(e) \geq \tau(q)$

- The ε-**maturity period** of q is a time period between
  - The first time stamp that $\sum_{\substack{\text{e arrives after q} \\ v(e) \in R(q)}} w(e) \geq (1-\varepsilon)\tau(q)$
  - The first time stamp that $\sum_{\substack{\text{e arrives after q} \\ v(e) \in R(q)}} w(e) \geq \tau(q)$

- Task: capture an arbitrary moment in the ε-**maturity period** of q

Problem Definition: Range Thresholding Problem

If there is only one query, this problem is easy.

[1] Mark de Berg et al. *Computational geometry: algorithms and applications, 3rd Edition*. Springer, 2008. ISBN: 9783540779735. URL: https://www.worldcat.org/oclc/227584184.

Problem Definition: Range Thresholding Problem

If there is only one query, this problem is easy.

The **challenge** lies in supporting a large number of queries simultaneously.

- We define
  - m: the number of queries
  - n : the number of elements in stream
- Brute force algorithm
  - Time complexity is $O(m \cdot n)$

[1] Mark de Berg et al. *Computational geometry: algorithms and applications, 3rd Edition*. Springer, 2008. ISBN: 9783540779735. URL: https://www.worldcat.org/oclc/227584184.

Problem Definition: Range Thresholding Problem

If there is only one query, this problem is easy.

The **challenge** lies in supporting a large number of queries simultaneously.

- We define
  - m: the number of queries
  - n : the number of elements in stream
- Brute force algorithm
  - Time complexity is $O(m \cdot n)$
- Stabbing based algorithms[1]
  - Time complexity still contains a term of $O((1 - \varepsilon) \cdot \tau_{max} \cdot m)$

All of the above algorithms cannot overcome **quadratic bounds**.

[1] Mark de Berg et al. *Computational geometry: algorithms and applications, 3rd Edition*. Springer, 2008. ISBN: 9783540779735. URL: https://www.worldcat.org/oclc/227584184.

The State of the Art Solution

- QGT[2] algorithm
  - The first *sub-quadratic* time complexity solution for RT problem
  - Time complexity:

$$O(m \cdot \log^{d+1} m \cdot \log \frac{1}{\varepsilon} + n \cdot \log^{d+1} m)$$

  - Space complexity:[3]

$$O(m_{\text{alive}} \cdot \log^{d} m_{\text{alive}})$$

[2] Miao Qiao, Junhao Gan, and Yufei Tao. "Range Thresholding on Streams". In: *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*. ACM, 2016, pp. 571–582.

[3] m_{alive} is the number of queries that are still running in the system

- Partition query with a Segment Tree

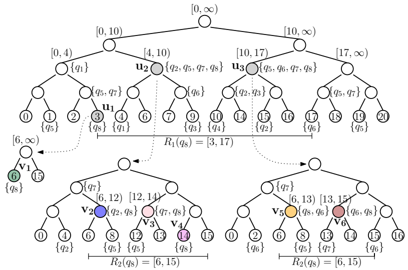

Figure: The Segment Tree on Q

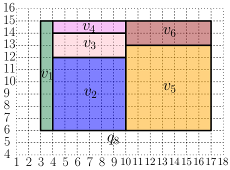

Figure: A partition of q_8

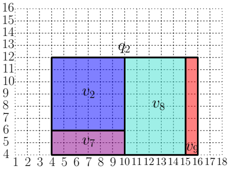

Figure: A partition of q_2

Basic idea of QGT

- Partition query with a Segment Tree
  - Multiple queries can **share** sub-range counters
- Track query maturity with Distributed Tracking Algorithm
  - **Distributed Tracking(DT)**[4] is a technique that helps to track when the sum of a set of counters reaches a certain threshold

[4] Graham Cormode, S. Muthukrishnan, and Ke Yi. "Algorithms for Distributed Functional Monitoring". In: *ACM Trans. Algorithms* 7.2 (Mar. 2011), 21:1–21:20. ISSN: 1549-6325.
[5] Jon Louis Bentley and James B. Saxe. "Decomposable Searching Problems I: Static-to-Dynamic Transformation". In: *J. Algorithms* 1.4 (1980), pp. 301–358.

Basic idea of QGT

- Partition query with a Segment Tree
  - Multiple queries can **share** sub-range counters
- Track query maturity with Distributed Tracking Algorithm
  - **Distributed Tracking(DT)**[4] is a technique that helps to track when the sum of a set of counters reaches a certain threshold
- Organize multiple DT instances with *Heap*
  - But will introduce $O(\log m)$ cost for each operation

[4] Graham Cormode, S. Muthukrishnan, and Ke Yi. "Algorithms for Distributed Functional Monitoring". In: *ACM Trans. Algorithms* 7.2 (Mar. 2011), 21:1–21:20. ISSN: 1549-6325.

[5] Jon Louis Bentley and James B. Saxe. "Decomposable Searching Problems I: Static-to-Dynamic Transformation". In: *J. Algorithms* 1.4 (1980), pp. 301–358.

Basic idea of QGT

- Partition query with a Segment Tree
  - Multiple queries can **share** sub-range counters
- Track query maturity with Distributed Tracking Algorithm
  - **Distributed Tracking(DT)**[4] is a technique that helps to track when the sum of a set of counters reaches a certain threshold
- Organize multiple DT instances with *Heap*
  - But will introduce $O(\log m)$ cost for each operation
- Support query insertion with *Logarithmic Method*[5]
  - Introduce $O(\log m)$ factor for the element and query processing time

[4] Graham Cormode, S. Muthukrishnan, and Ke Yi. "Algorithms for Distributed Functional Monitoring". In: *ACM Trans. Algorithms* 7.2 (Mar. 2011), 21:1–21:20. ISSN: 1549-6325.

[5] Jon Louis Bentley and James B. Saxe. "Decomposable Searching Problems I: Static-to-Dynamic Transformation". In: *J. Algorithms* 1.4 (1980), pp. 301–358.

Limitations of QGT

The limitations of QGT algorithm:

- Utilize **Heap** to organize multiple DT instances
  - $O(\log m)$ running time overhead for DT instances processing
- Utilize **Logarithmic Method** to support query dynamic
  - $O(\log m)$ running time overhead for element and query processing
- **Space consumption** in practice
  - Run out of 100GB memory for 2 million 3-dimensional queries
  - Our algorithm on the same dataset only uses 10GB memory to give the result within 3 hours

Limitations of QGT

The limitations of QGT algorithm:

- Utilize **Heap** to organize multiple DT instances
  - $O(\log m)$ running time overhead for DT instances processing
- Utilize **Logarithmic Method** to support query dynamic
  - $O(\log m)$ running time overhead for element and query processing
- **Space consumption** in practice
  - Run out of 100GB memory for 2 million 3-dimensional queries
  - Our algorithm on the same dataset only uses 10GB memory to give the result within 3 hours

We aim to design an algorithm that is fast and space-efficient in **practice** with sub-quadratic **theoretical** bound.

Our Solution: FastRTS

Our Solution: FastRTS

Comparing to QGT, FastRTS eliminates:

- The *Heap* with **Bucketing Technique**
- The *Logarithmic Method* with **Incremental Segment Tree**

[6] N is the size of the universe $\mathbb{U}$ on each dimension

Our Solution: FastRTS

Comparing to QGT, FastRTS eliminates:

- The *Heap* with **Bucketing Technique**
- The *Logarithmic Method* with **Incremental Segment Tree**

Our FastRTS algorithm achieves

- Time complexity in expectation:[6]

$$O(m \cdot \log^d N \cdot \log \frac{1}{\varepsilon} + n \cdot \log^d N)$$

- Space complexity:

$$O(m_{\text{alive}} \cdot \log^d N)$$

[6] N is the size of the universe $\mathbb{U}$ on each dimension

Our Solution: FastRTS

Table: The complexity comparison

Algorithms	Overall Running Time Cost	Space Consumption
FastRTS	$O(m \cdot \log^{d} N \cdot \log \frac{1}{\varepsilon} + n \cdot \log^{d} N)$ expected	$O(m_{\text{alive}} \cdot \log^{d} N)$
QGT algorithm	$O(m \cdot \log^{d+1} m \cdot \log \frac{1}{\varepsilon} + n \cdot \log^{d+1} m)$	$O(m_{\text{alive}} \cdot \log^{d} m)$

Our Solution: FastRTS

Table: The complexity comparison

Algorithms	Overall Running Time Cost	Space Consumption
FastRTS	$O(m \cdot \log^{d} N \cdot \log \frac{1}{\varepsilon} + n \cdot \log^{d} N)$ expected	$O(m_{\text{alive}} \cdot \log^{d} N)$
QGT algorithm	$O(m \cdot \log^{d+1} m \cdot \log \frac{1}{\varepsilon} + n \cdot \log^{d+1} m)$	$O(m_{\text{alive}} \cdot \log^{d} m)$

Some notes on N:

- In practice scenarios, N is usually **not very large**.
  - Prices range is usually bounded within million of cents.
- We propose two **effective optimizations** to reduce the dependency on N in practice.
  - Even N is as large as 10^9, our performance is still quite stable.

Organize DT with Bucketing Technique

Eliminate *Heap* with *Bucketing Technique*:

- Propose a new DT algorithm: *Power-of-Two-Slack DT*
- Organize *Power-of-Two-Slack DT* with a linked list of buckets
- $O(1)$ expected time complexity, which reduces a logarithmic factor for DT processing.

Support Query Dynamics with Incremental Segment Tree

Eliminate *Logarithmic Method* with *Incremental Segment Tree*:

- Maintain a Segment Tree on the whole universe $\mathbb{U}^d$
- Only materialize the nodes touched by alive queries
- Support query dynamic easily

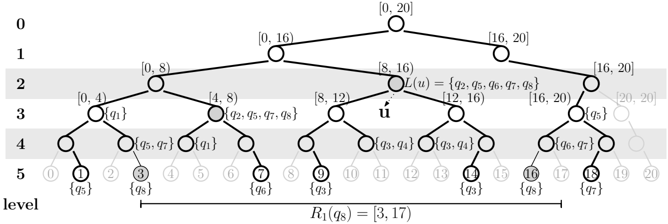

Figure: *Example of IncSegTree* on the first dimension

Optimizations on FastRTS

That alone is not enough to make FastRTS run fast enough in practice.

Optimizations on FastRTS

That alone is not enough to make FastRTS run fast enough in practice.

Two powerful optimizations for FastRTS:

- The Range Shrinking Technique
- The Range Counting Technique

Optimizations on FastRTS

That alone is not enough to make FastRTS run fast enough in practice.

Two powerful optimizations for FastRTS:

- The Range Shrinking Technique
- The Range Counting Technique

Advantages

- Reduce the actual running time and peak memory usage
- Retain all the theoretical bounds

The Range Shrinking Technique

- Extend the query range to its *super range* to get a *Super Query*.
  - Cost for tracking *Super Query* is significantly small
  - q never misses the maturity moment before its *Super Query* matures
- Benefits:
  - Have a chance of *early stop*
  - Keep the DT instance small most of time and make the peak memory usage of each DT instance asynchronous

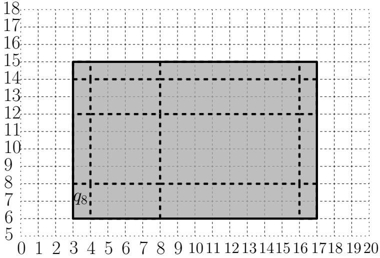

Figure: Original query q_8

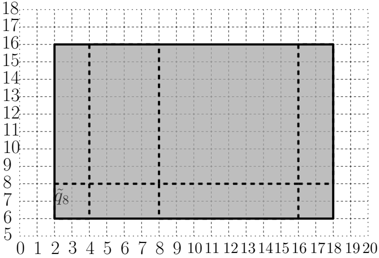

Figure: Super query $\tilde{q}_8$

The Range Counting Technique

- Drawback of Range Shrinking Technique:
  - Still need to materialize too much nodes
  - Otherwise, we lose precise counter information
- Support precise counter collection with Range Tree[7].
- Benefits:
  - Only need to materialize the nodes touched by *Super Query* of q

[7] Mark de Berg et al. *Computational geometry: algorithms and applications, 3rd Edition*. Springer, 2008. ISBN: 9783540779735. URL: https://www.worldcat.org/oclc/227584184.

Experiment on Synthetic Data

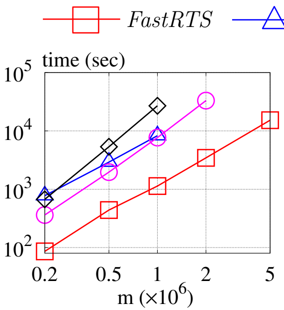

Figure: Overall Running Time

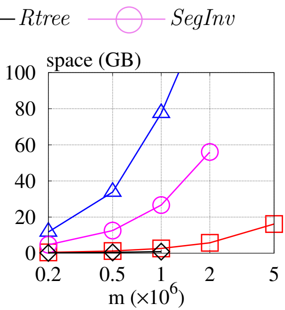

Figure: Peak Memory Usage

- $d = 3$, $\varepsilon = 0.05$, $\tau = m$, $n = 20 \cdot m$, $N = 10$ million

Experiment on Real Stock Trading Data

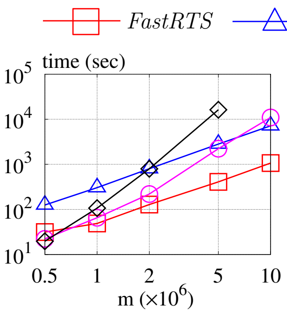

Figure: Overall Running Time

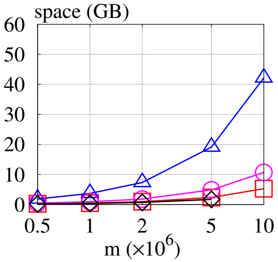

Figure: Peak Memory Usage

- $d = 2$, $\varepsilon = 0.05$, $\tau = m$, $n = 20 \cdot m$, $N = 100,000$

Thanks