**KLEE: A Framework for Distributed Top-k Query Algorithms**

**Sebastian Michel**  
Max-Planck Institute for Informatics  
Saarbrücken, Germany  
smichel@mpi-inf.mpg.de

**Peter Triantafillou**  
RACTI / Univ. of Patras  
Rio, Greece  
peter@ceid.upatras.gr

**Gerhard Weikum**  
Max-Planck Institute for Informatics  
Saarbrücken, Germany  
weikum@mpi-inf.mpg.de
Overview

- Problem Statement
- Related Work
- KLEE
- The Histogram Bloom Structure
- Candidate Filtering
- Evaluation
- Conclusion / Future Work
Computational Model

- Distributed aggregation queries: *Query with m terms with index lists spread across m peers P1 ... Pm*

Applications:
- Internet traffic monitoring
- Sensor networks
- P2P Web search
Problem Statement

Query initiator P0 serves as per-query coordinator

• Consider
  – network consumption
  – per peer load
  – latency (query response time)
    • network
    • I/O
    • processing
Related Work

Existing Methods:

- **Distributed NRA/TA**: Extend NRA/TA (Fagin et al. `99/’03, Güntzer et al. `01, Nepal et al. `99) with batched access

- **TPUT (Cao/Wang 2004)**:
  1. fetch k best entries \((d, s_j)\) from each of P1 \(\ldots\) Pm and aggregate \((\sum_{j=1}^{m} s_j(d))\) at P0
  2. ask each of P1 \(\ldots\) Pm for all entries with \(s_j > \min-k/m\) and aggregate results at P0
  3. fetch missing scores for all candidates by random lookups at P1 \(\ldots\) Pm

+ DNRA aims to minimize per-peer work
- DTA/DNRA incur many messages
+ TPUT guarantees fixed number of message rounds
- TPUT incurs high per-peer load and net BW
KLEE: A Framework for Distributed Top-k Query Algorithms
KLEE: Key Ideas

• If $\frac{m_{\text{ink}}}{m}$ is small TPUT retrieves a lot of data in Phase 2
  ➔ high network traffic

• random accesses
  ➔ high per-peer load

KLEE:

- Different philosophy: approximate answers!
- Efficiency:
  - Reduces (docId, score)-pair transfers
  - no random accesses at each peer
- Two pillars:
  - The HistogramBlooms structure
  - The Candidate List Filter structure
The KLEE Algorithms

• KLEE 3 or 4 steps:

1. Exploration Step: … to get a better approximation of min-k score threshold

2. Optimization Step:
   – decide: 3 or 4 steps?

3. Candidate Filtering: … a docID is a good candidate if high-scored in many peers.

4. Candidate Retrieval: get all good docID candidates.
Histogram Bloom Structure

- Each peer pre-computes for each index list:
  - an equi-width histogram
  - a Bloom filter for each cell
  - average score per cell
  - upper/lower score

"increase" the mink / m threshold
Bloom Filter

- bit array of size $m$
- $k$ hash functions
  $$h_i: \text{docId\_space} \rightarrow \{1,..,m\}$$
- insert $n$ docs by hashing the ids and settings the corresponding bits
- **Membership Queries:**
  - document is in the Bloom Filter if the corresponding bits are set
- probability of false positives ($pfp$)
  $$pfp = (1 - e^{-kn/m})^k$$
- tradeoff accuracy vs. efficiency
Exploration and Candidate Retrieval

Coordinator Peer P0

Current top-k

Candidate set

Cohort Peer Pi

Index List

Candidate set

Histogram

b bits

c bold

min-k / m

Coordinator Peer Pj

Index List

Candidate set

Histogram

b bits

c bold

min-k / m
Candidate List Filter Matrix

• Goal: filter out unpromising candidate documents in step 2
• estimate the max number of docs that are above the min_k / m threshold

• send this number and the threshold to the cohort peers
Candidate List Filter Matrix (2)

- Each cohort returns a Bloom Filter that “contains” all docs above the $\text{mink} / \text{m}$ threshold

**Candidate List Filter Matrix (CLFM)**

<table>
<thead>
<tr>
<th>010101001011110101001001010101001</th>
</tr>
</thead>
<tbody>
<tr>
<td>01001001100101111100100101011110</td>
</tr>
<tr>
<td>101010101010100110010010011110000</td>
</tr>
<tr>
<td>101010101010100110010010011110000</td>
</tr>
</tbody>
</table>

Select all columns with at least R bits set
KLEE– Candidate Set Reduction

Coordinator Peer P0

min-k / m

current top-k

candidate set

Cohort Peer Pi

top k

candidates

score

Index List

Candidate filter matrix

min-k / m

Cohort Peer Pj

00001000010000001

00010000010000001

00001000010000001

00001000010000001

00010000010000001

00001000010000001

00001000010000001

00001000010000001

00001000010000001

00001000010000001

00001000010000001

00001000010000001

00001000010000001
KLEE – Candidate Retrieval

Coordinator Peer P0

current top-k candidate set

Index List

score

candidate filter matrix

min-k / m
current top-k
candidate set
candidate set

Cohort Peer Pi
top k candidates

Cohort Peer Pj

early stopping point

010010000100010001

0000100000100000001

0000100000100000001

0000100000100000001

100010100000010001

0000100000100000001

0000100000100000001
Enhanced Filtering

• BF representation can be improved …

<table>
<thead>
<tr>
<th>d1</th>
<th>d2</th>
<th>d5</th>
<th>d3</th>
<th>d4</th>
<th>d17</th>
<th>d9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9</td>
<td>0.6</td>
<td>0.5</td>
<td>0.3</td>
<td>0.25</td>
<td>0.08</td>
<td>0.07</td>
</tr>
</tbody>
</table>

• Send byte-array with cell-numbers instead of bits

• Select „columns“ with

  Sum over upper-bounds > min-k

...
KLEE: A Framework for Distributed Top-k Query Algorithms

Architecture/Testbed

1. open()
2. get(k)
3. getAbove(score)
4. getWithBF(,.)

open() next() close()

SQL

B+ Index

Extended IndexLists with BloomFilters, Histograms, and Batched Access

Index Lists

Oracle DB

KLEE Algorithmic Framework
Evaluation: Benchmarks

- **GOV**: TREC .GOV collection + 50 TREC-2003 Web queries, e.g. *juvenile delinquency*
- **XGOV**: TREC .GOV collection + 50 manually expanded queries, e.g. *juvenile delinquency youth minor crime law jurisdiction offense prevention*
- **IMDB**: Movie Database, queries like
  - actor = John Wayne; genre =western
- **Synthetic Distribution** (Zipf, different skewness): GOV collection but with synthetic scores
- **Synthetic Distribution + Synthetic Correlation**: 10 index lists
Evaluation: Metrics

- Relative recall w.r.t. to the actual results
- Score error
- Bandwidth consumption
- Rank distance
- Number of RA and number of SA
- Query response time
  - network cost (150ms RTT, 800Kb/s data transfer rate)
  - local I/O cost (8ms rotation latency + 8MB/s transfer delay)
  - processing cost
Evaluated Algorithms

- **DTA:**
  - batched distributed threshold algorithm, batch size k.

- **TPUT**

- **X-TPUT:**
  - approximate TPUT. No random accesses.

- **KLEE-3**
  - C = 10% of the score mass

- **KLEE-4**
## Synthetic Score Benchmarks

<table>
<thead>
<tr>
<th>Zipf-GOV</th>
<th>Total # of Bytes</th>
<th>Total Time in ms</th>
<th>Average Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTA</td>
<td>17,752,769</td>
<td>3,532,180</td>
<td>1</td>
</tr>
<tr>
<td>TPUT</td>
<td>53,494,903</td>
<td>576,713</td>
<td>1</td>
</tr>
<tr>
<td>X-TPUT</td>
<td>53,011,252</td>
<td>404,991</td>
<td>0.99</td>
</tr>
<tr>
<td>KLEE 3</td>
<td>49,861,342</td>
<td>367,931</td>
<td>0.97</td>
</tr>
<tr>
<td>KLEE 4</td>
<td>25,057,920</td>
<td>160,585</td>
<td>0.94</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Zipf-XGOV</th>
<th>Total # of Bytes</th>
<th>Total Time in ms</th>
<th>Average Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTA</td>
<td>617,009,260</td>
<td>39,582,682</td>
<td>1</td>
</tr>
<tr>
<td>TPUT</td>
<td>377,928,880</td>
<td>1,599,581</td>
<td>1</td>
</tr>
<tr>
<td>X-TPUT</td>
<td>377,097,644</td>
<td>1,521,220</td>
<td>0.98</td>
</tr>
<tr>
<td>KLEE 3</td>
<td>287,294,812</td>
<td>1,189,891</td>
<td>0.91</td>
</tr>
<tr>
<td>KLEE 4</td>
<td>165,077,807</td>
<td>375,077</td>
<td>0.92</td>
</tr>
</tbody>
</table>

### Zipf-XGOV, $c=10\%$, $\theta=0.7$

![Bandwidth Bar Chart](chart.png)

$\theta = 0.7$
KLEE: A Framework for Distributed Top-k Query Algorithms

VLDB 2005, Trondheim

Synthetic Correlation Benchmark

<table>
<thead>
<tr>
<th>Overlap+Zipf c=10% ( \Omega = 30% )</th>
<th>Total # of Bytes</th>
<th>Total Time in ms</th>
<th>Average Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTA</td>
<td>1,146.32</td>
<td>157,420</td>
<td>1</td>
</tr>
<tr>
<td>TPUT</td>
<td>9,150,904</td>
<td>29,270</td>
<td>1</td>
</tr>
<tr>
<td>X-TPUT</td>
<td>9,150,904</td>
<td>28,335</td>
<td>1</td>
</tr>
<tr>
<td>KLEE 3</td>
<td>3,678,780</td>
<td>12,971</td>
<td>0.92</td>
</tr>
<tr>
<td>KLEE 4</td>
<td>1,192,704</td>
<td>6,546</td>
<td>0.91</td>
</tr>
</tbody>
</table>

randomly insert top \( k \) documents from list \( i \) in the top \( \Omega \) documents of list \( j \)
GOV / XGOV

<table>
<thead>
<tr>
<th>GOV, c=10%</th>
<th>Total # of Bytes</th>
<th>Total Time in ms</th>
<th>Average Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTA</td>
<td>1,172,446</td>
<td>190,259</td>
<td>1</td>
</tr>
<tr>
<td>TPUT</td>
<td>1,505,290</td>
<td>185,049</td>
<td>1</td>
</tr>
<tr>
<td>X-TPUT</td>
<td>597,991</td>
<td>31,432</td>
<td>0.89</td>
</tr>
<tr>
<td>KLEE 3</td>
<td>722,664</td>
<td>28,319</td>
<td>0.9</td>
</tr>
<tr>
<td>KLEE 4</td>
<td>440,868</td>
<td>39,564</td>
<td>0.9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>XGOV, c=10%</th>
<th>Total # of Bytes</th>
<th>Total Time in ms</th>
<th>Average Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTA</td>
<td>92,587,264</td>
<td>3,740,677</td>
<td>1</td>
</tr>
<tr>
<td>TPUT</td>
<td>70,044,884</td>
<td>2,346,882</td>
<td>1</td>
</tr>
<tr>
<td>X-TPUT</td>
<td>19,236,084</td>
<td>96,153</td>
<td>0.91</td>
</tr>
<tr>
<td>KLEE 3</td>
<td>16,690,912</td>
<td>88,271</td>
<td>0.83</td>
</tr>
<tr>
<td>KLEE 4</td>
<td>7,920,774</td>
<td>56,609</td>
<td>0.79</td>
</tr>
</tbody>
</table>
• Conclusion
  – KLEE: approximate top-k algorithms for wide-area networks
  – significant performance benefits can be enjoyed, at only small penalties in result quality
  – flexible framework for top-k algorithms, allowing for trading-off
    • efficiency versus result quality and
    • bandwidth savings versus the number of communication phases.
  – various fine-tuning parameters

• Future Work
  – Reasoning about parameter values
  – Consider “moving” coordinator
Thanks for your attention!

Questions?

Comments?