



max planck institut
informatik

Optimizing Imprecise Aggregation Queries in Large Distributed Networks

Thomas Neumann

Max-Planck-Institut Informatik, Saarbrücken, Germany

October 16, 2008

Motivation

example: distributed multimedia archiving with retrieval

- data is distributed over peers
- organized by feature, each peer computes approximate score
- use this as pre-filter, re-check the final candidates

node1 (feature 1): node2 (feature 2): node3 (feature 3):

document	score	document	score	document	score
img123	0.9	img345	0.5	img333	1.0
img345	0.7	img123	0.4	img123	0.3
img888	0.4	img111	0.3	img345	0.1

Return the top- k documents that have the highest aggregated scores.

(somewhat simplified, in reality we want to minimize L_1)



Motivation (2)

For distributed top- k queries

- network access is the limiting factor
- centralized algorithms not applicable
- specialized algorithms reduce network accesses
- have fixed evaluation schemes, though

Our approach

- improve decentralized algorithms by optimization techniques
- exploit data and network characteristics to improve performance
- use biased sampling if necessary

Goal: minimize query response times



Overview

1. Existing Approaches
2. Optimization Techniques
3. Evaluation
4. Conclusion



Existing Approaches - TPUT

Cao and Wang [PODC04]: Three Phase Uniform Threshold (TPUT)

Base execution strategy:

1. Exploration Step

- request the top k items from all n nodes
- aggregate, compute (partial) score of k -th best item (min_k)

2. Candidate Retrieval

- request all items with a score $\geq \frac{min_k}{n}$ from all nodes
- aggregate, compute worst-score and best-score

3. Missing Score Lookup

- request missing scores to get exact order
- update the aggregates, compute the final top k

Note: fixed retrieval, adapts neither to data nor network characteristics.



Existing Approaches - KLEE

Fast approximate algorithm [VLDB05]

- exchanges histograms to get better (higher) min_k estimates
- constructs Bloom filters with all items above min_k to filter early
- leaves out random lookups at the end

Speeds up processing, but still a fixed scheme.



Optimization Techniques - Adaptive Thresholds

Observation: The scan threshold of TPUT/KLEE is quite arbitrary

- we must see all items with total score $> \min_k$ at least once
- using $t = \frac{\min_k}{n}$ guarantees this
- but could use different t_i for different peers
- as long as $\sum_{i=1}^n t_i \leq \min_k$ we are safe

Rough Idea: Increase t_i for large peers, reduce for small peers

Will reduce number of transferred items. But how to choose t_i ?



Optimization Techniques - Adaptive Thresholds (2)

- in a slightly loaded network we wait for the slowest peer
 - choose the t_i to minimize the maximum scan depth d
1. guess a value for d
 2. for each peer P_i , set t_i to the maximum score such that d items are above t_i
 3. check if $\sum_{i=1}^n t_i \leq \min_k$
 4. if not, increase d , otherwise decrease d (binary search)

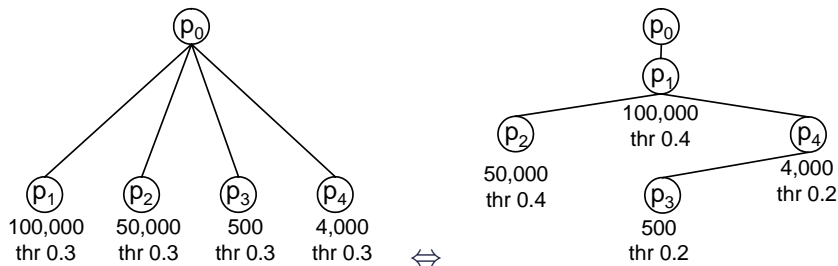
We use score histograms to estimate the correct t_i .



Optimization Techniques - Trees

- adaptive thresholding describes how to aggregate peers
- exploits data characteristics but not network characteristics

Aggregation can be done hierarchically:



Reduces data early, avoids large transfers



Optimization Techniques - Trees (2)

Find optimal hierarchy using dynamic programming:

buildHierarchy(L, min_k)

Input: set L of all data-item lists; value threshold min_k

Output: optimal execution plan

if (l, min_k) has already been solved **then return** known solution

b = flat adaptive aggregation of L , threshold min_k

if $|L| > 1$ **then**

for each $P = \{L_i \subset L\}$, P partitioning of L

$L' = \{\text{buildHierarchy}(L_i, min_k/|P|) \mid L_i \in P\}$

a = aggregation of L'

if $a.costs < b.costs$ **then** $b = a$

store b as solution for (L, min_k) in DP table

return b

simplified version, ignores explicit data transfers

Optimization Techniques - Sampling

Optimizations help a lot, but not enough for hundreds of peers:

- thresholds nearly zero, we have to read everything

Instead, examine only a sample:

- use biased sampling, prefer nodes with a high score mass
- sample until we have seen $x\%$ of the score mass

Extra problem here: statistical synopsis

- all optimizations require estimates for score distribution etc.
- but with sampling we cannot ask them!
- instead, integrate tiny histograms in peer index



Evaluation

Our multimedia data set is not complete yet (project ongoing), we tested with typical IR settings for now.

Setup:

- Internet-style topology, ns-2 network simulator

Algorithms:

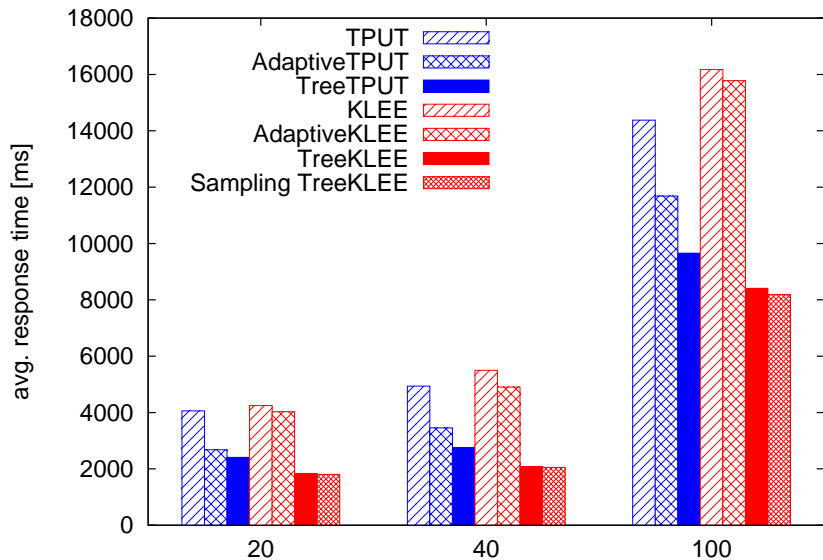
- TPUT and KLEE, with adaptive thresholding only (Adaptive*) and full optimization (Tree*)
- sampling on top of TreeKLEE

Data sets:

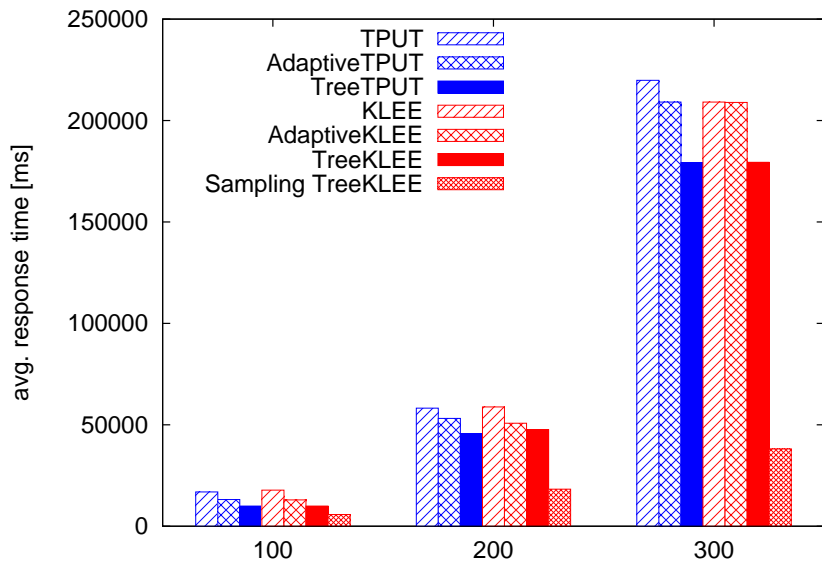
- Retail:
 - $\sim 88k$ transactions from Belgian retail store
 - created 100 peers (distributed shopping sites)
 - search frequent triplets
- AOL:
 - 20M queries from AOL



Evaluation - Retail



Evaluation - AOL



Evaluation - Recall

Recall for approximate evaluation:

#Peers	TPUT	Adaptive TPUT	Tree TPUT	KLEE	Adaptive KLEE	Tree KLEE	Sampling TreeKLEE
Retail							
40	0.98	0.97	0.97	0.92	0.90	0.90	0.85
100	0.98	0.96	0.96	0.92	0.90	0.90	0.85
AOL							
200	0.99	0.98	0.98	0.99	0.98	0.98	0.90
300	0.99	0.97	0.97	0.98	0.97	0.97	0.89

- TPUT/KLEE can be exact by final lookup phase
- optimizations affect recall a bit due to seeing less (very minor)
- sampling reduces recall, but not that much and improves runtimes a lot



Conclusion

Results:

- existing top- k algorithms do not exploit data and network properly
- **adaptive thresholding** adapt retrieval to data distribution
- hierarchical **aggregation trees** adapt retrieval to the network
- combination of both greatly improves performance
- **sampling** not that important for 'small' queries (< 100 peers), but essential for larger ones

Future work:

- integrate optimizations into other algorithms
- include correlations in our statistical models

