

Algebraic Query Optimization for Distributed Top-k Queries

Thomas Neumann

Sebastian Michel

March 8, 2007

Overview

1. Motivation
2. Base Algorithm: TPUT
3. Algebraic Formulation
4. Query Optimization
5. Evaluation
6. Conclusion

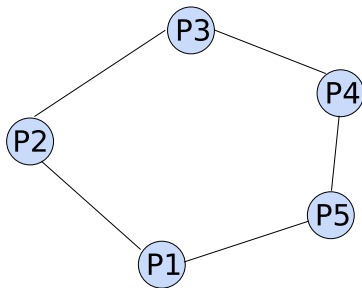
Motivation:

Distributed aggregation queries: Query with n terms with index lists spread across n Peers P_1, \dots, P_n

Applications:

- ▶ Internet traffic monitoring
- ▶ Sensor networks
- ▶ P2P Web search

Find the "best" k results.



Motivation (2)

A typical distributed top- k queries (in SQL-like notation):

```
select    id, sum(score)
from      (part1 union part2 union ... union part $n$ )
group by id
order by sum(score) desc
limit     k
```

- ▶ part fragments are distributed
- ▶ computing the union explicitly is very expensive
- ▶ note that the aggregation can be more complex than **sum**

Motivation (3)

Problems:

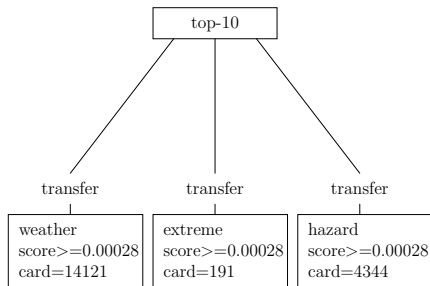
- ▶ in the distributed setting data access is expensive
- ▶ thus, the computation itself has to be distributed
- ▶ latency is usually high, avoid round trips
- ▶ rules out most centralized top- k algorithms

Existing solutions:

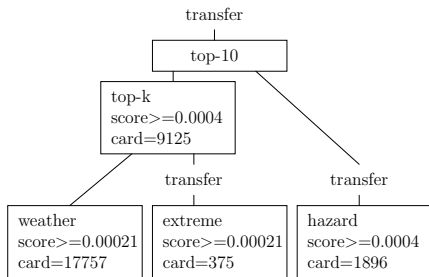
- ▶ range scans on (relevant) scores
- ▶ fixed execution scheme, algorithms do not change per query
- ▶ data flow etc. not optimized per query

Motivation (4)

Two equivalent execution plans:



standard execution
2580 ms



hierarchical execution
689 ms

Hierarchical execution more powerful, but:

- ▶ identical result must be guaranteed, performance gain estimated

Base Algorithm: 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: items are pruned due to worst-score/best-score comparisons

Base Algorithm: TPUT (2)

Threshold choice is conservative:

- ▶ the final top k will have a score of at least min_k
- ▶ if a item as a score of less than $\frac{min_k}{n}$ on all nodes it will never make it into the top k
- ▶ thus all relevant tuples are seen

But computation can be improved:

- ▶ data could be **aggregated** earlier
- ▶ computation could be **moved** to different nodes

We provide optimization techniques for both.

Algebraic Formulation

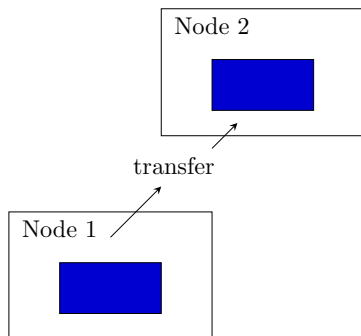
- ▶ to make TPUT more flexible, break it into smaller parts
- ▶ each part is provided by an algebraic operator
- ▶ operators allow for both a more flexible data flow and moving the computation
- ▶ standard operator model from relational algebra

Note that we only optimize Phase 2, i.e. min_k is known.

Algebraic Formulation - Transfer

Most basic operation for distributed queries: **transfer**

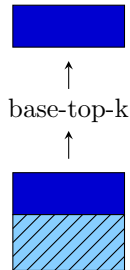
- ▶ sends data to another node
- ▶ data itself not modified
- ▶ causes latency and bandwidth delay



Algebraic Formulation - Base Top k

Leaf operator in top- k processing: **base-top-k**

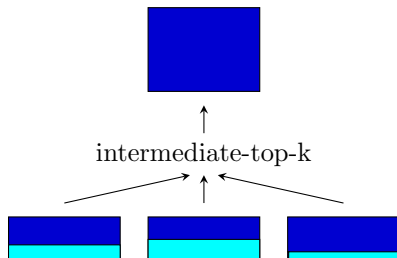
- ▶ scans data fragment
- ▶ produces all candidates above threshold
- ▶ data still local



Algebraic Formulation - Intermediate Top k

Intermediate operator: **intermediate-top-k**

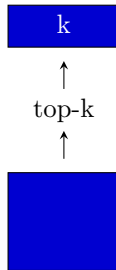
- ▶ aggregates candidate sets
- ▶ merges duplicates
- ▶ prunes with new bounds against threshold
- ▶ produces new candidate set



Algebraic Formulation - Top k

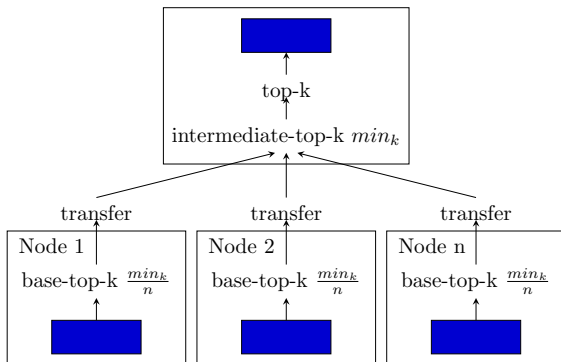
Final operator: **top-k**

- ▶ considers candidate list
- ▶ retrieves missing scores
- ▶ prunes candidates
- ▶ constructs final top k



Algebraic Formulation - TPUT

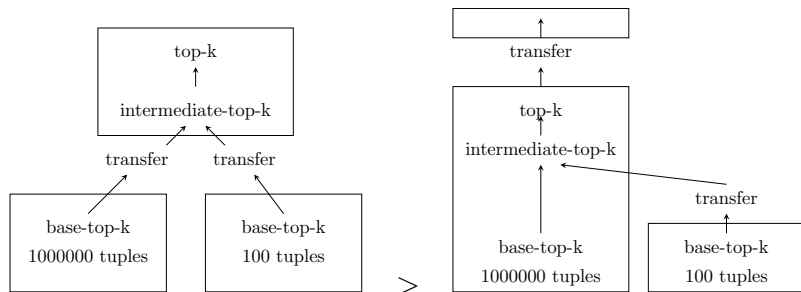
Using these operator rules, TPUT can be formulated as:



- ▶ in fact TPUT is a special case of our generalized execution strategy

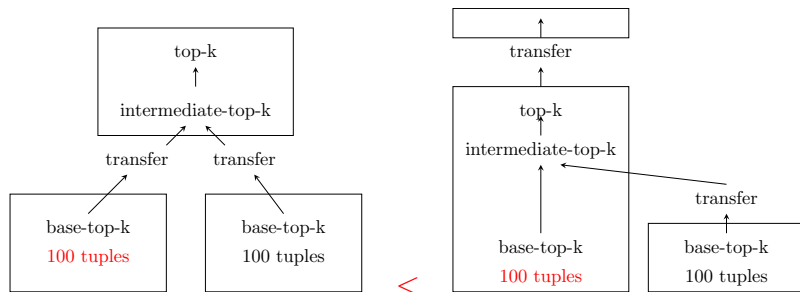
Query Optimization - Simple

- ▶ moving the computation does not affect the result
- ▶ computing near the data is usually a good idea



Query Optimization - Simple

- ▶ moving the computation does not affect the result
- ▶ computing near the data is usually a good idea
- ▶ but not always



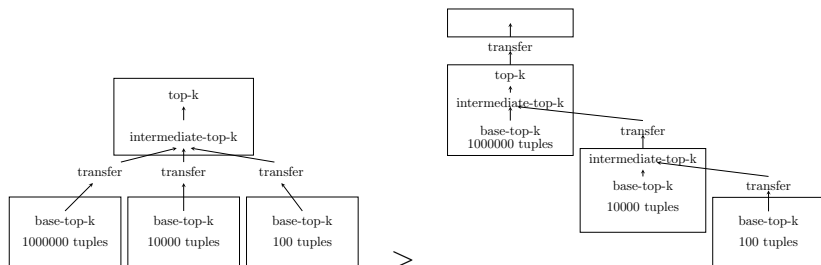
- ▶ query optimization decision

Query Optimization - Simple (2)

- ▶ determine the best operator placement
- ▶ change transfers accordingly
- ▶ cheap to compute
- ▶ avoids the worst mistakes
- ▶ but does not aggregate hierarchical

Query Optimization - Optimal

Aggregation can be rearranged, too.



- ▶ prediction quite involved (thresholds \rightarrow cardinalities)

Query Optimization - Optimal (2)

Find optimal plan with DP:

- ▶ for each aggregation, consider optimal aggregations for all partitionings
- ▶ consider transfers at the same time

Gain:

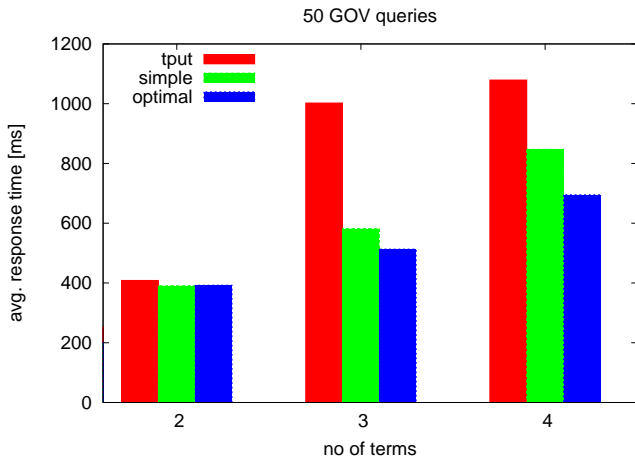
- ▶ considers much broader class of execution plans
- ▶ TPUT and *simple* are special cases
- ▶ returns the optimal hierarchical execution plan

Evaluation

Two IR data sets with queries:

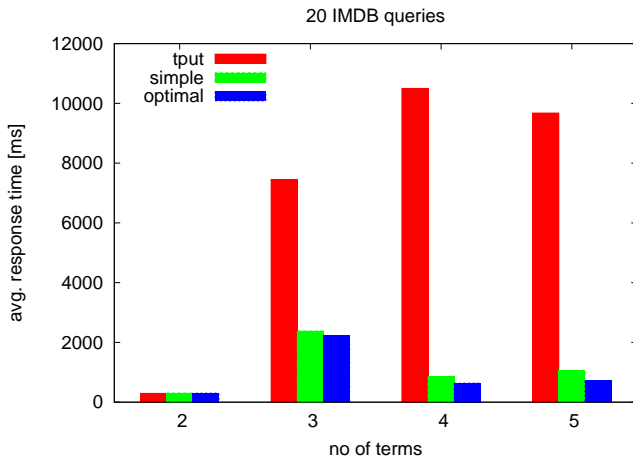
- ▶ GOV: TREC-12 Web Track, ca. 1.25 million tuples
- ▶ IMDB: Internet Movie Database, ca. 1.6 million tuples
- ▶ only show response times here
- ▶ results grouped by number of query terms (i.e. number of data fragments)
- ▶ different benchmark queries operate on different partitions, though
- ▶ absolute values not directly comparable between queries

Evaluation - GOV



- ▶ difference between *simple* and *optimal* grows with number of fragments

Evaluation - IMDB



- ▶ note that *optimal* is better than *simple*! (nearly factor 2 for 5)

Conclusion

- ▶ algebraic formulation of distributed top- k processing
- ▶ aggregate data, move computation
- ▶ easy gains from moving computation
- ▶ hierarchical computation even better
- ▶ more complex queries allow for more optimization
- ▶ query optimization greatly improved base algorithm