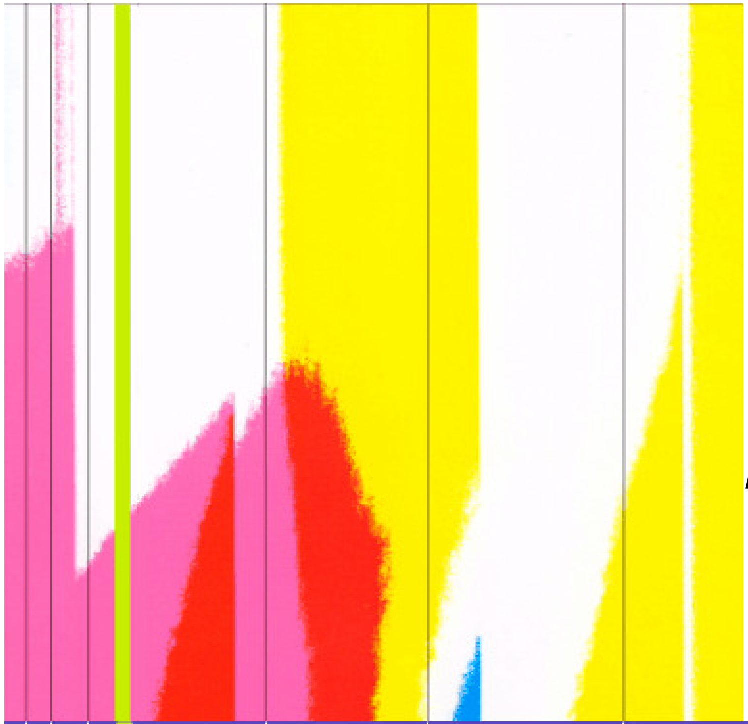


Best Position Algorithms for Top-k Queries*

Reza Akbarinia, Esther Pacitti, Patrick Valduriez
Atlas Team, INRIA and LINA, Nantes

September 2007

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

* **VLDB Conference, 2007**

INRIA
RENNES

UNIVERSITÉ DE NANTES

lina
LABORATOIRE D'INFORMATIQUE
DE NANTES ATLANTIQUE
CNRS FRE 2729
Université de Nantes
Ecole des Mines de Nantes

Outline

Introduction

Problem Definition

Related Work (FA and TA)

Best Position Algorithm (BPA)

Optimization (BPA2)

Performance Evaluation

Conclusion

Top-k Query

Returns only the *k* most relevant answers

- Scoring function (*sf*): determines the answers' relevance (*score*)

Advantage: avoid overwhelming the user with large numbers of uninteresting answers

Useful in many areas

- Network and system monitoring
- Information retrieval
- Multimedia databases
- Sensor networks
- Data stream systems
- P2P systems
- Etc.

Hard to support efficiently

- Need to aggregate overall scores from local scores

General Model for Top-k Queries [Fagin99]

Suppose we have:

- *n* data items
- *m* lists of the *n* data items such that
  - Each data item has
    - a local score in each list
  - Each list
    - is sorted in decreasing order of the local scores
- Overall score of a data item: computed based on its local scores in all lists using a given scoring function

The objective is:

- *Find the k data items whose overall scores are the highest w.r.t. a given scoring function*

General Model - illustration

Top-k tuples in relational tables:

- Have a sorted list (index) over each attribute
- Then, find the *k* tuples whose overall scores in the lists are the highest

Top-k documents wrt. some given keywords:

- Have for each keyword, a ranked list of documents
- Then, find the *k* documents whose overall scores in the lists are the highest

Execution Cost of Top-k Algorithms

Calculated based on the accesses to the lists

Two types of access to the lists [FLN01]

- Sorted (sequential) access (SA)
  - Reads next item in the list (starts with the first data item)
- Random access (RA)
  - Looks up a given data item in the list by its identifier (e.g. TID)

Execution cost of a top-k algorithm *A* over a database *D (i.e. set of sorted lists)* is:

$$Cost(A, D) = (\text{num\_SA} \times \text{cost\_SA}) + (\text{num\_RA} \times \text{cost\_RA})$$

Problem Definition

Assumption:

- Scoring function is monotonic, i.e. $sf(x) \leq sf(y)$ if $x < y$
  - Many of the popular aggregation functions are monotonic, e.g. Sum, Min, Max, Avg, …

Given

- m lists of n data items (also called a database)
- A monotonic scoring function
- An integer k such that $k \leq n$

Objective:

- Find the k data items whose overall score is the highest, while minimizing execution cost

Related Work

Fagin's Algorithm (FA) [Fagin, JCSS99]

- A simple algorithm
  - Do sorted access in parallel to the lists until *at least k data items have been seen in all lists*

Threshold Algorithm (TA)

- The most efficient algorithm (so far) over sorted lists
- The basis for many TA-style distributed algorithms
- Proposed independently by several groups
  - [Nepal and Ramakrishna, ICDE99]
  - [Fagin, Lotem and Naor, PODS01]
  - [Güntzer, Kießling and Balke, ITCC01]

TA

Similar to FA in doing sorted access to the lists, but with a different stopping condition:

- After seeing each data item, TA does random access to other lists to read the data item’s score in all lists
- It uses a *threshold (T)* to predict maximum possible score of unseen items
  - Based on the last scores seen in the lists under sorted access
- It stops when there are at least *k* seen data items whose overall score ≥ T

TA Example

sf () = $s_1 + s_2 + s_3$, k = 3

Y: {top seen items}

Y = {(b, 70), (c, 70), (a, 65)}

Threshold ≤ score of k items: then stop

T=30+28+30 = 88

T=28+27+29 = 84

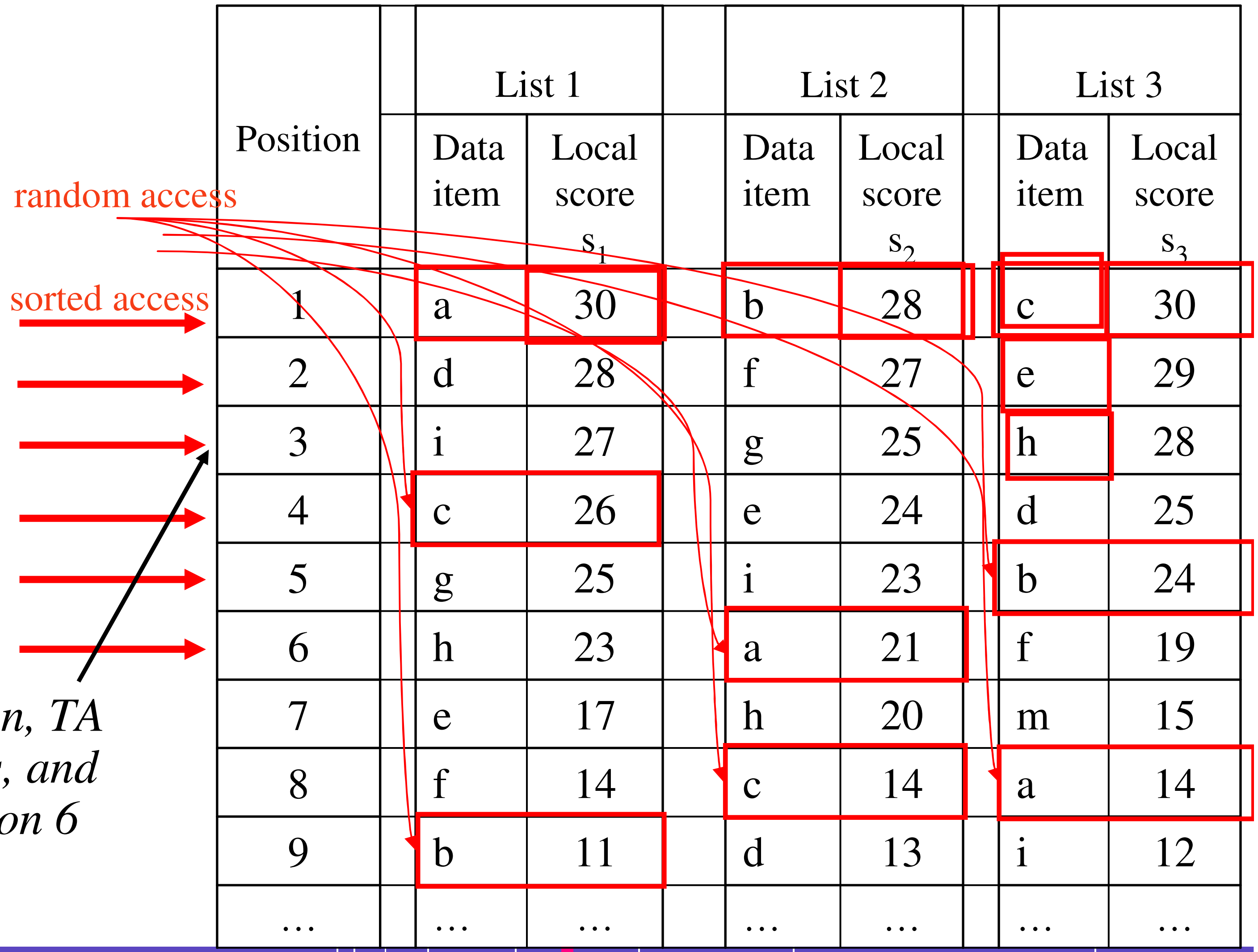
T=27+25+28 = 80

T=26+24+25 = 75

T=25+23+24 = 72

T=23+21+19 = 63

But at the 3rd position, TA has all top-k answers, and continues until position 6



random access

sorted access

Position	List 1 Data item	List 1 Local score s_1	List 2 Data item	List 2 Local score s_2	List 3 Data item	List 3 Local score s_3
1	a	30	b	28	c	30
2	d	28	f	27	e	29
3	i	27	g	25	h	28
4	c	26	e	24	d	25
5	g	25	i	23	b	24
6	h	23	a	21	f	19
7	e	17	h	20	m	15
8	f	14	c	14	a	14
9	b	11	d	13	i	12
…	…	…	…	…	…	…

Best Position Algorithm (BPA)

Main idea: *take into account the positions (and scores) of the seen items for stopping condition*

- Enables BPA to stop much sooner than TA

Best position = the greatest seen position in a list such that any position before it is also seen

- Thus, we are sure that all positions between 1 and *best position* have been seen

Stopping condition

- Based on *best positions overall score*, i.e. the overall score computed based on the best positions in all lists

BPA

Do sorted access in parallel to each list L_i

- For each data item seen in L_i
  - Do random access to the other lists to retrieve the item's score and position
  - Maintain the positions and scores of the seen data item
- Compute *best position* in L_i
- Compute *best positions overall score*
- Stop when there are at least *k* data items whose overall score ≥ *best positions overall score*

BPA Example

sf () = $s_1 + s_2 + s_3$, k = 3

Y: {top seen items}

Y = {(c, 70), (a, 65), (b, 63)}

Y = {(h, 71), (e, 70), (c, 70)}

Y = {(c, 70), (e, 70), (a, 65)}

Best Positions:

Best Positions Overall Score =

30 + 28 + 30 = 88

Best Positions:

Best Positions:

Best Positions Overall Score =

28 + 27 + 29 = 84

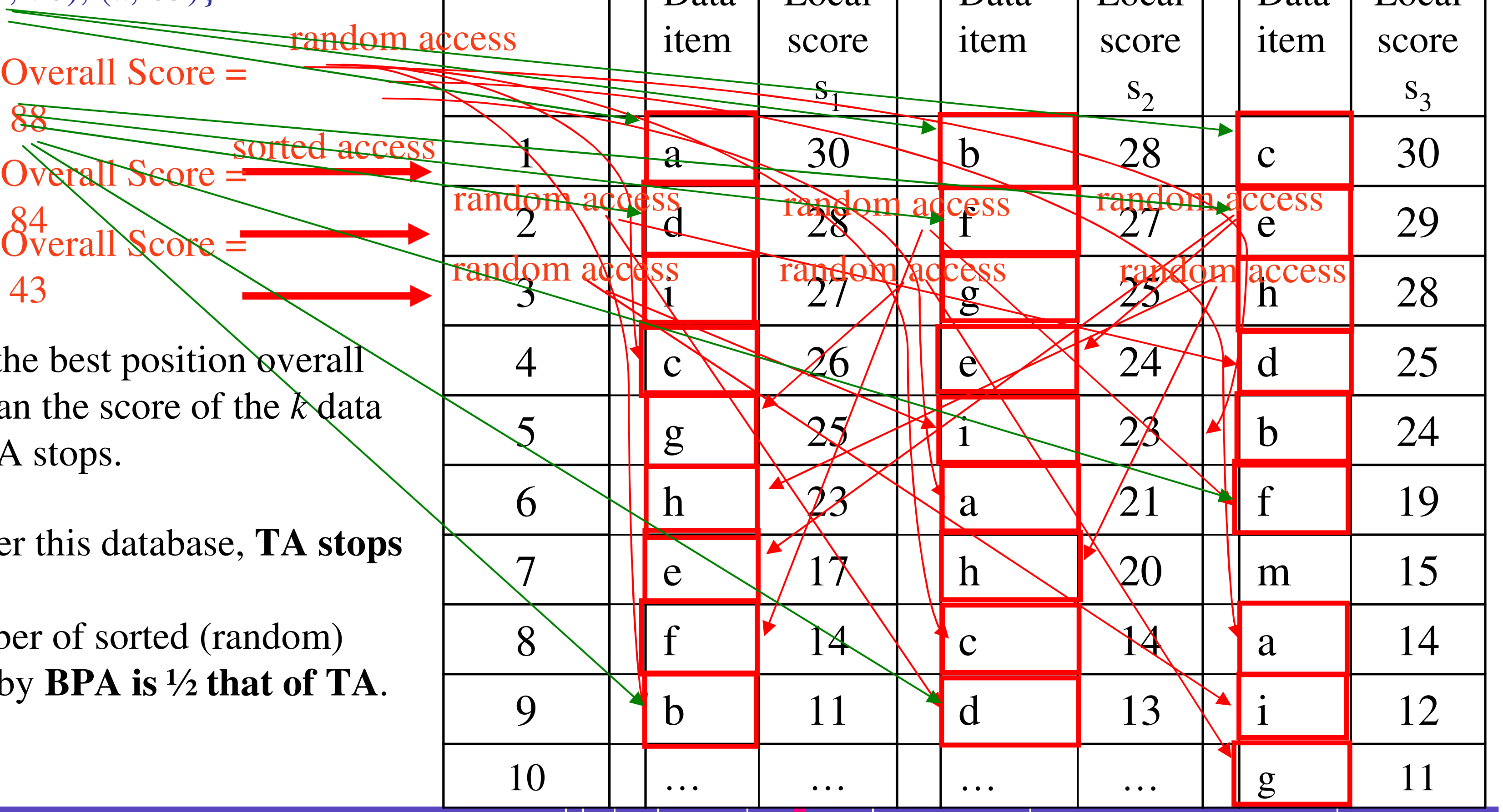
Best Positions Overall Score =

11 + 13 + 19 = 43

At position **3**, the best position overall score is less than the score of the *k* data items, thus BPA stops.

Recall that, over this database, **TA stops at position 6**.

Thus, the number of sorted (random) accesses done by **BPA is ½ that of TA**.



Position	List 1		List 2		List 3	
	Data item	Local score s_1	Data item	Local score s_2	Data item	Local score s_3
1	a	30	b	28	c	30
2	d	28	f	27	e	29
3	i	27	g	25	h	28
4	c	26	e	24	d	25
5	g	25	i	23	b	24
6	h	23	a	21	f	19
7	e	17	h	20	m	15
8	f	14	c	14	a	14
9	b	11	d	13	i	12
10	…	…	…	…	g	11

BPA Analysis

Lemma 1. The number of sorted (random) accesses done by BPA is always less than or equal to that of TA. In other words, BPA stops always as early as TA.

Theorem 1. The execution cost of BPA over any database is always less than or equal to that of TA.

Theorem 2. The execution cost of BPA can be *(m-1)* times lower than that of TA, where *m* is the number of lists.

BPA Optimization: BPA2

Main optimizations

- Uses the *direct access* mode
  - Retrieves the data item which is at a given position in a list
- Avoids re-accessing data via sorted or random access
  - In BPA, a data item may be accessed several times in different lists
  - In BPA2, no data item in a list is *accessed more than once*
- Manages best positions of a list by
  - Bit array or B+-tree over the list

BPA2

For each list L_i do in parallel

- Let bp_i be the best position in L_i. Initially set $bp_i=0$
- Continually do direct access to position ($bp_i + 1$)
  - Do random access to the other lists to retrieve the scores of the seen data item in all lists
  - After each direct or random access to a list, update the best position of the list
- Stop when there are at least k data items whose overall score ≥ *best positions overall score*

Analysis of BPA2

Theorem 3. No position in a list is accessed by BPA2 more than once.

Theorem 4. The number of accesses to the lists done by BPA2 can be approximately *(m-1)* times lower than that of BPA.

Performance Evaluation

Implementation of TA, BPA and BPA2

- To study the performance in the average case

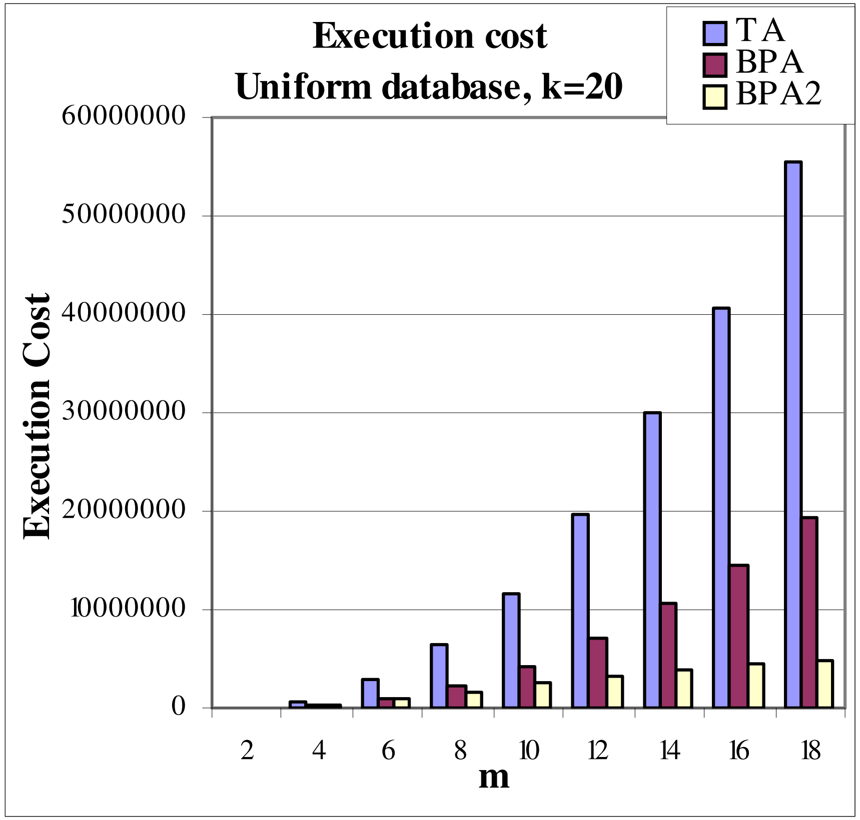
Synthetic data sets

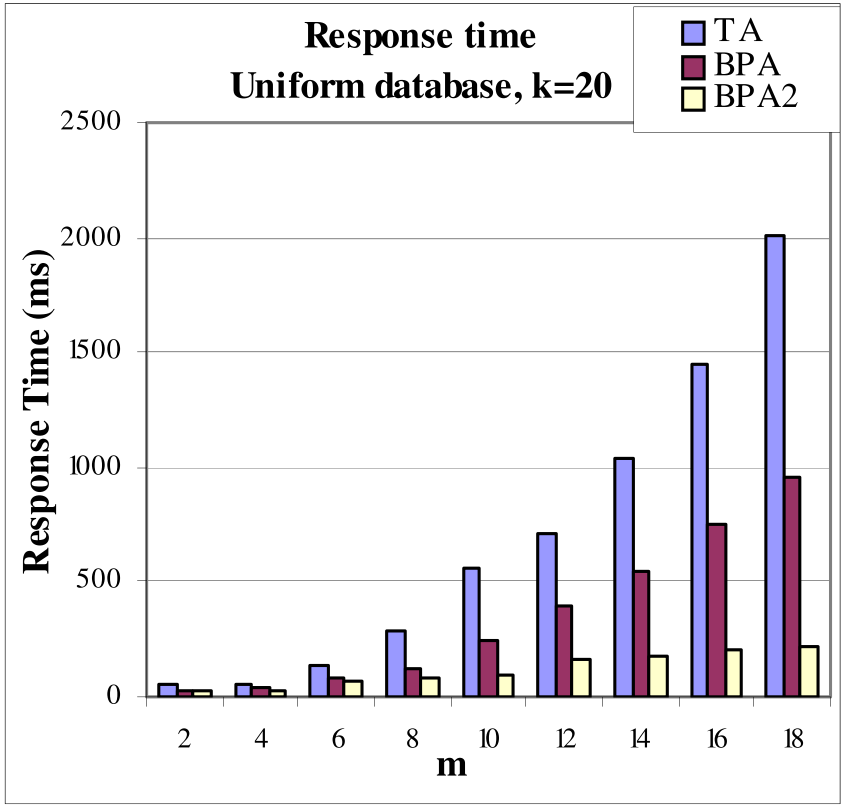
- Uniform
- Gaussian
- Correlated

Metrics

- Execution cost
  - Customized for centralized systems
  - Cost of a random access is (log n) times of a sorted access
- Number of accesses
  - Useful in distributed systems
- Response time
  - Over a machine with a 2.4 GHz Intel Pentium 4

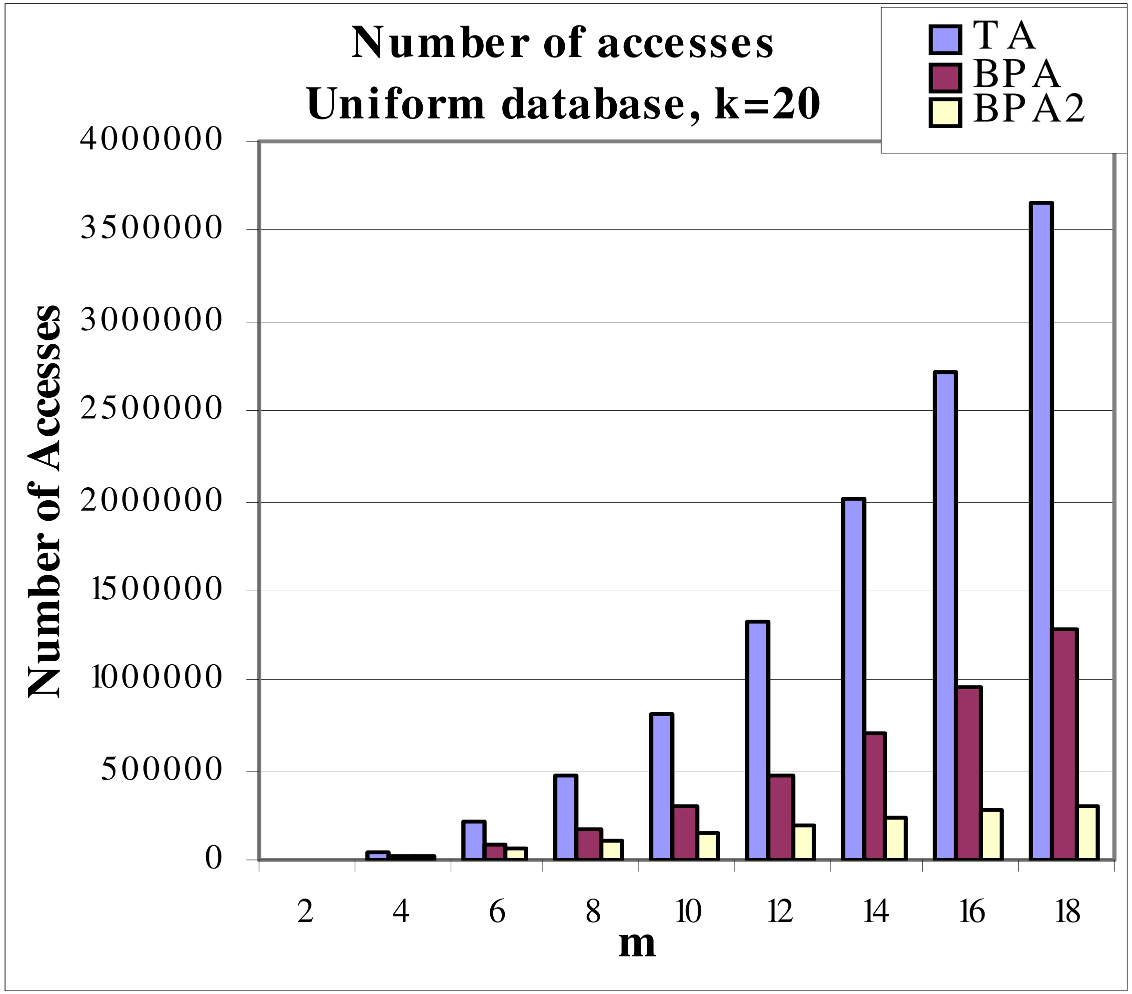
Response Time and Execution Cost vs. Number of Lists





BPA and BPA2 outperform TA by a factor of about *(m/8 + 0.75)* and *(m/2 +0.5)* respectively (for *m*>2).

Number of Accesses vs. Number of Lists



Conclusion

BPA

- Over any database, it stops as early as TA
- Its execution cost can be (m-1) times lower than that of TA

BPA2

- Avoids re-accessing data items via sorted and random access, without having to keep data at the query originator
- The number of accesses to the lists done by BPA2 can be about *(m-1)* times lower than that of BPA

Validation and performance evaluation

- BPA and BPA2 outperform TA by significant factors

Future Work

- BPA-style algorithms for P2P systems, in particular for DHTs

Thank You
Merci

Questions ?

References

R. Akbarinia, E. Pacitti, P. Valduriez. Best Position Algorithms for Top-k Queries. *In Proc. of VLDB Conf.,* pages 495-506, 2007.

R. Akbarinia, E. Pacitti, P. Valduriez. Data Currency in Replicated DHTs. *In Proc. of ACM SIGMOD Conf.*, pages 211-222, 2007.

R. Akbarinia, E. Pacitti and P. Valduriez. Processing Top-k Queries in Distributed Hash Tables. *In Proc. of Euro-Par Conf.*, pages 489-502 , 2007.

R. Akbarinia, E. Pacitti and P. Valduriez. Reducing network traffic in unstructured P2P systems using top-k queries. *Journal of Distributed and Parallel Databases*, 19(2-3), pages 67-86, 2006.

R. Akbarinia and V. Martins. Data management in the APPA P2P system. *Journal of Grid Computing*, 5(3), pages 303-317, 2007.

R. Akbarinia, V. Martins, E. Pacitti, and P. Valduriez. Design and implementation of APPA. *Global Data Management* (Eds. R. Baldoni, G. Cortese and F. Davide), IOS Press, 2006.

R. Akbarinia, V. Martins, E. Pacitti and P. Valduriez. Top-k query processing in the APPA P2P system. *In Proc. of the Int. Conf. on High Performance Computing for Computational Science (VecPar)*, LNCS 4395, Springer, pages 158-171, 2006.

R. Akbarinia, E. Pacitti and P. Valduriez. An efficient mechanism for processing top-k queries in DHTs. *In Proc. of the Journées Bases de Données Avancées (BDA)*, 2006.

R. Akbarinia, V. Martins, E. Pacitti and P. Valduriez. Replication and query processing in the APPA data management system. *In Proc. of the Int. Workshop on Distributed Data and Structures (WDAS)*, Carleton Scientific, pages 19-33, 2004.

References

[Fag99] R. Fagin. Combining fuzzy information from multiple systems. *J. Comput. System Sci., 58 (1)*, 1999.

[FLN01] R. Fagin, A. Lotem and M. Naor. Optimal aggregation algorithms for middleware. *PODS Conf.*, 2001.

[GKB00] U. Güntzer, W. Kießling and W.-T Balke. Optimizing multi-feature queries for image databases. VLDB Conf., 2000.

[NR99] S. Nepal and M.V. Ramakrishna. Query processing issues in image (multimedia) databases. *ICDE Conf.*, 1999.

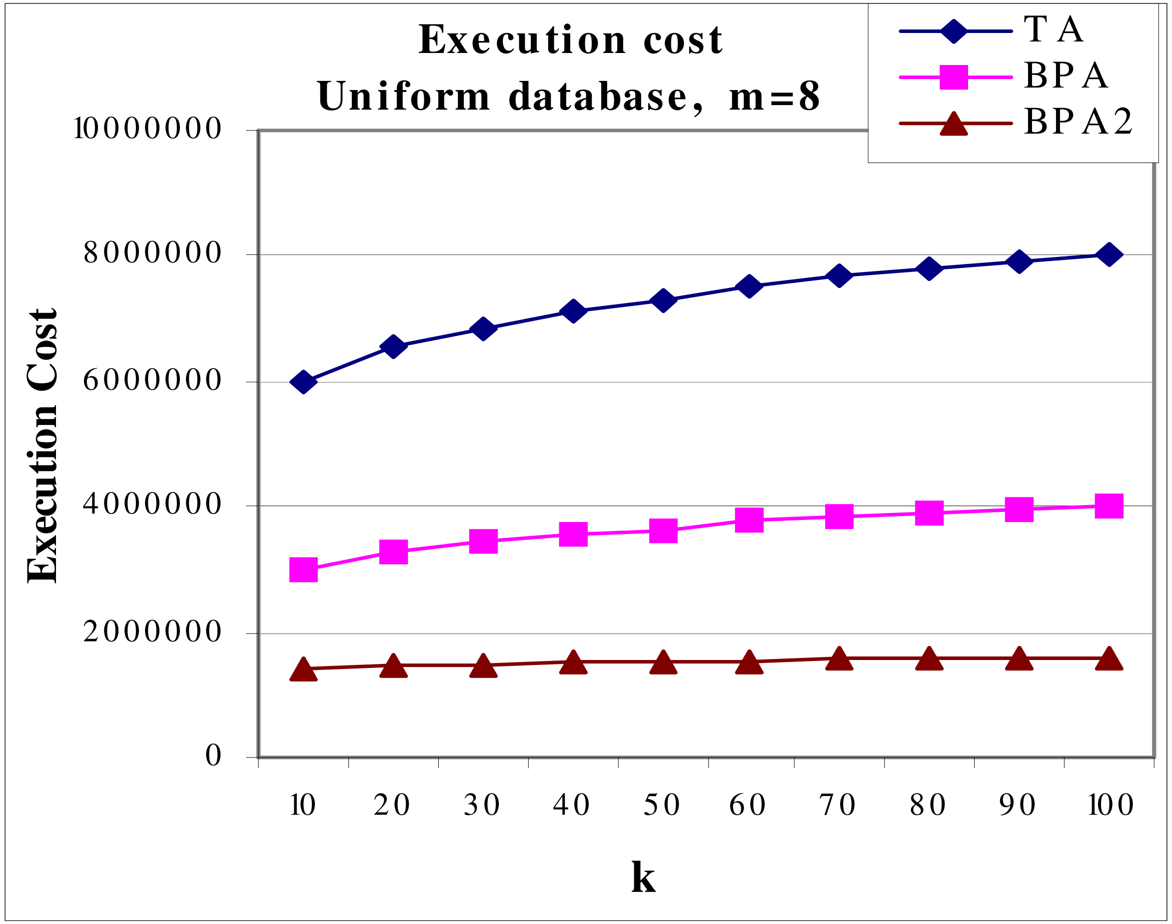
FAQ

Are there applications in which we need a large number of lists (i.e. m >> 1) ?

Example : A network monitoring application

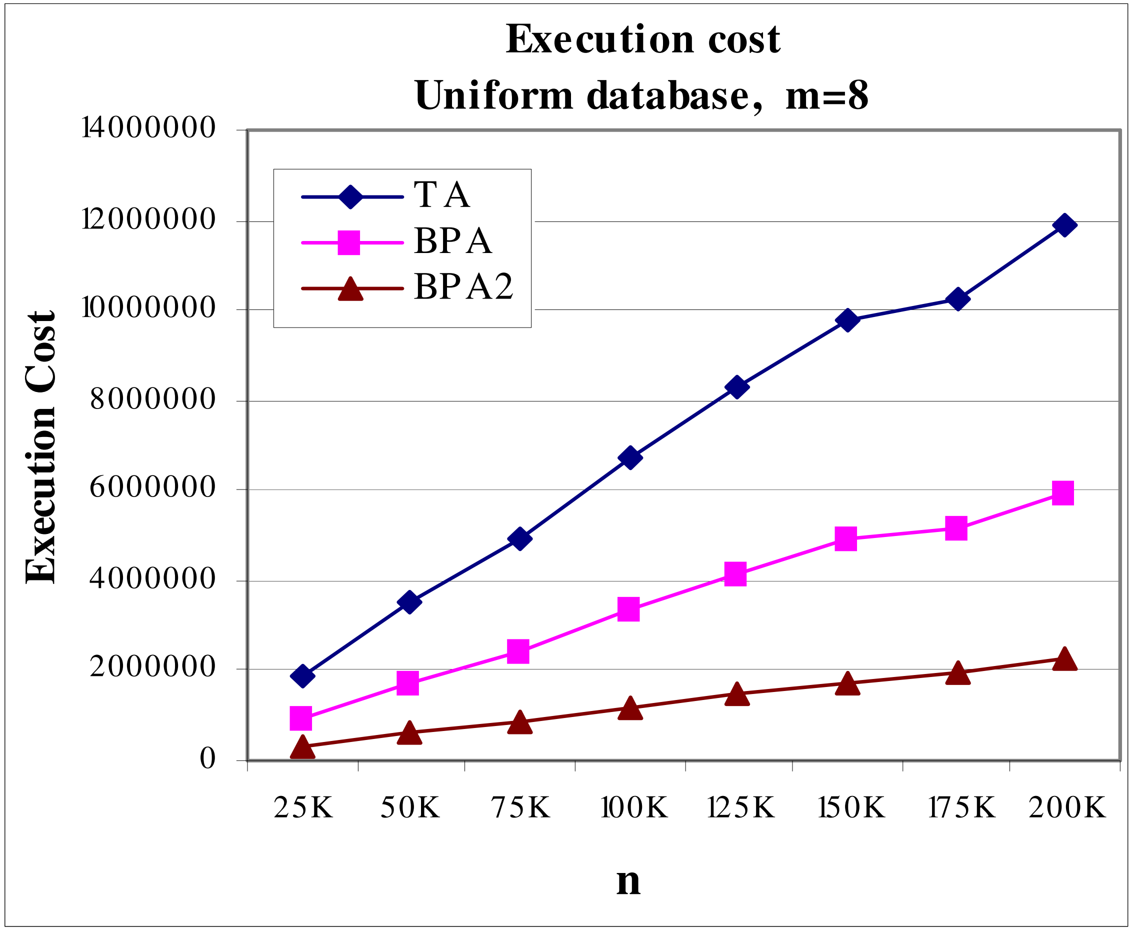
- It monitors the activities of the users of some specified IP locations
- The specified locations may be numerous (e.g. > 1000)
- For each location, the application maintains a list of the accessed URLs ranked by their frequency of access
- Query: what are the top-k popular URLs accessed by the locations?

Execution Cost vs. *k*



R1

Effect of the Number of Data Items



Folie 27

R1 Units

Reza; 20.09.2007