On Saying "Enough Already!" in MapReduce

Christos Doulkeridis and Kjetil Nørvåg

Department of Computer and Information Science (IDI)
Norwegian University of Science and Technology (NTNU)
Trondheim, Norway

NTNU
Norwegian University of
Science and Technology

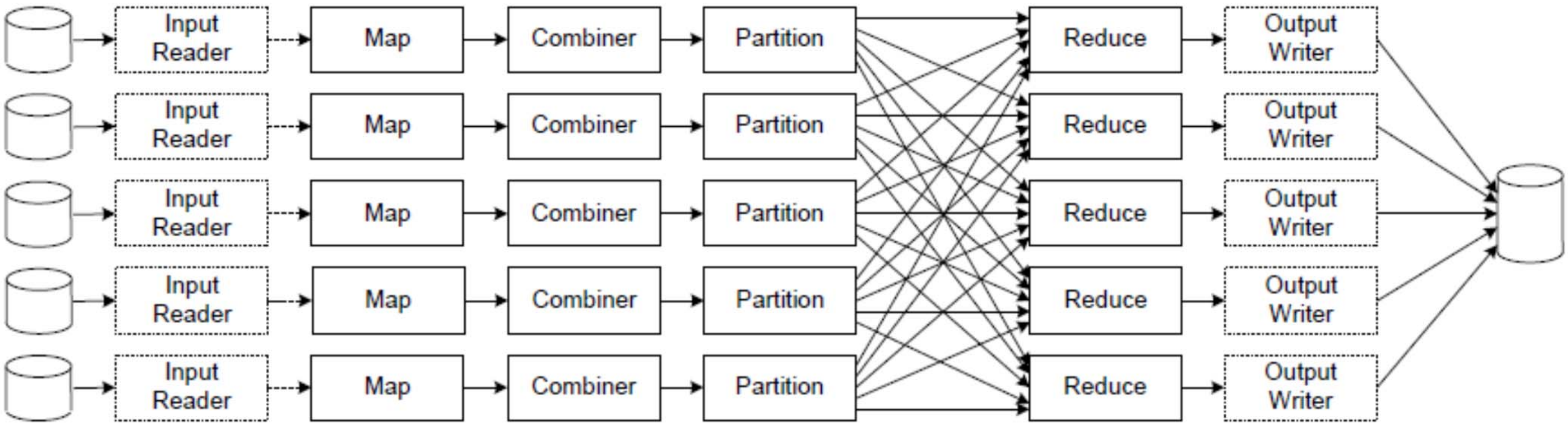
Outline

- Motivation
- Preliminaries
  - MapReduce
  - Top-k queries and top-k joins
- Rank-aware query processing in MapReduce
  - Sorted access
  - Intelligent data placement
  - Data synopses
- Related work
- Conclusions and outlook

Motivation

- *Business intelligence* (*BI*) technology is essential for effective decision-making for the enterprise
- The challenges and needs of BI applications explode in the *era of "Big Data"*
  – Data collection from various sources
    - Retail, banking, RFID tags, email, query logs, blogs, reviews, etc.
- *Cloud Intelligence* for data analysis of massive data sets
  – The only scalable solution to-date
- Popularity of *MapReduce* and its open-source implementation *Hadoop*
  – Important to support *advanced BI operators over MapReduce*
  – Focus of this work: *efficient rank-aware query processing*

MapReduce – Overview



Map: $(k_1, v_1) \rightarrow [(k_2,v_2)]$
Reduce: $(k_2, [v_2]) \rightarrow [v_3]$

- Salient features
  - Scalability, fault-tolerance, ease of use, flexibility, …
- Limitations
  - Performance !!!
  - Lack of early termination

Rank-aware Processing

- Important tool for BI applications
  - Decision-making based on top-k results
    - matching the user's constraints, and
    - ranked according to user preferences
  - Inspection of a bounded set of k tuples only
    - Rather than retrieval and display of a huge result set
  - Challenging to report the top-k result, without exhaustive access to the underlying input data (*early termination*)

Top-k Queries

SELECT *

FROM hotels

WHERE

hotels.city=“Trondheim”

ORDER BY hotels.price

STOP AFTER *k*;

ID	NAME	CITY	PRICE
h1	BEST	Trondheim	1400
h2	ABC	London	1200
h3	HOLIDAY	Trondheim	1500
h4	CHEAP	Trondheim	1200
h5	HILTON	London	1800

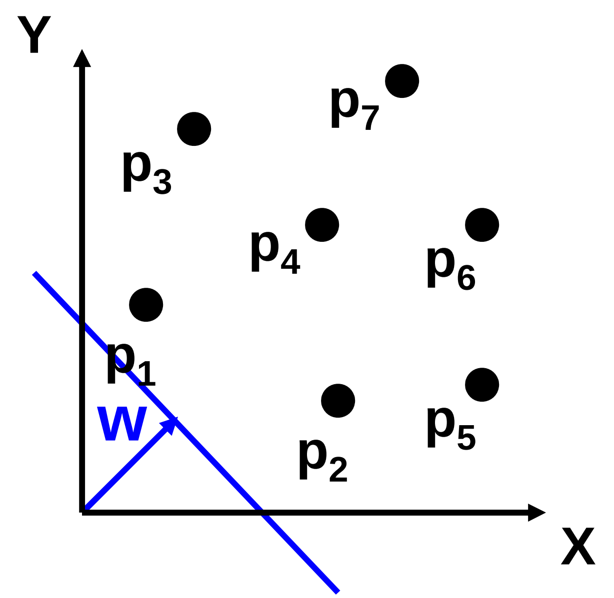
h1	BEST	Trondheim	1400
h3	HOLIDAY	Trondheim	1500
h4	CHEAP	Trondheim	1200

$k = 1$

h4	CHEAP	Trondheim	1200
h1	BEST	Trondheim	1400
h3	HOLIDAY	Trondheim	1500

Top-k Queries

- A top-k query $q_k(f)$ is defined by a **user-specified scoring function *f***, which
  - aggregates the objects' characteristics into a single score
    - *E.g.,* $f = w_1 * X + w_2 * Y$ where $\Sigma w_i = 1$
  - defines a total ordering



- Given
  - a positive integer *k and*
  - a user-defined weighting vector ***w***

  Find
  - the *k* data points *p* with the minimum *f(p)* scores

Top-k Queries

SELECT *
FROM hotels **HOLIDAY**
WHERE
hotels.city = “Trondheim”
ORDER BY
0.5*hotels.**price**+**0.5***hotels.**dist**
STOP AFTER *1*;

SELECT *
FROM hotels **CHEAP**
WHERE
hotels.city = “Trondheim”
ORDER BY
0.9*hotels.**price**+**0.1***hotels.**dist**
STOP AFTER *1*;

ID	NAME	CITY	**PRICE**	**DIST**	SCORE
h1	BEST	Trondheim	1400	2000	1700 / 1460
h2	ABC	London	1200	3000	
h3	HOLIDAY	Trondheim	1500	1500	1500
h4	CHEAP	Trondheim	1200	2200	1700 / 1300
h5	HILTON	London	1800	800	

Top-k Joins

SELECT *
FROM hotels, flights
WHERE
hotels.city = flights.to_city
ORDER BY
(**0.5***hotels.price + **0.5***flights.price)
STOP AFTER *k*

ID	NAME	CITY	PRICE
h1	BEST	Trondheim	1400
h2	ABC	London	1200
h3	HOLIDAY	Trondheim	1500
h4	CHEAP	Trondheim	1200
h5	HILTON	London	1800

ID	AIRLINE	TO_CITY	PRICE
f1	KLM	Trondheim	5000
f2	KLM	London	4200
f3	SAS	Trondheim	3000
f4	SAS	Trondheim	3500
f5	KLM	London	2000

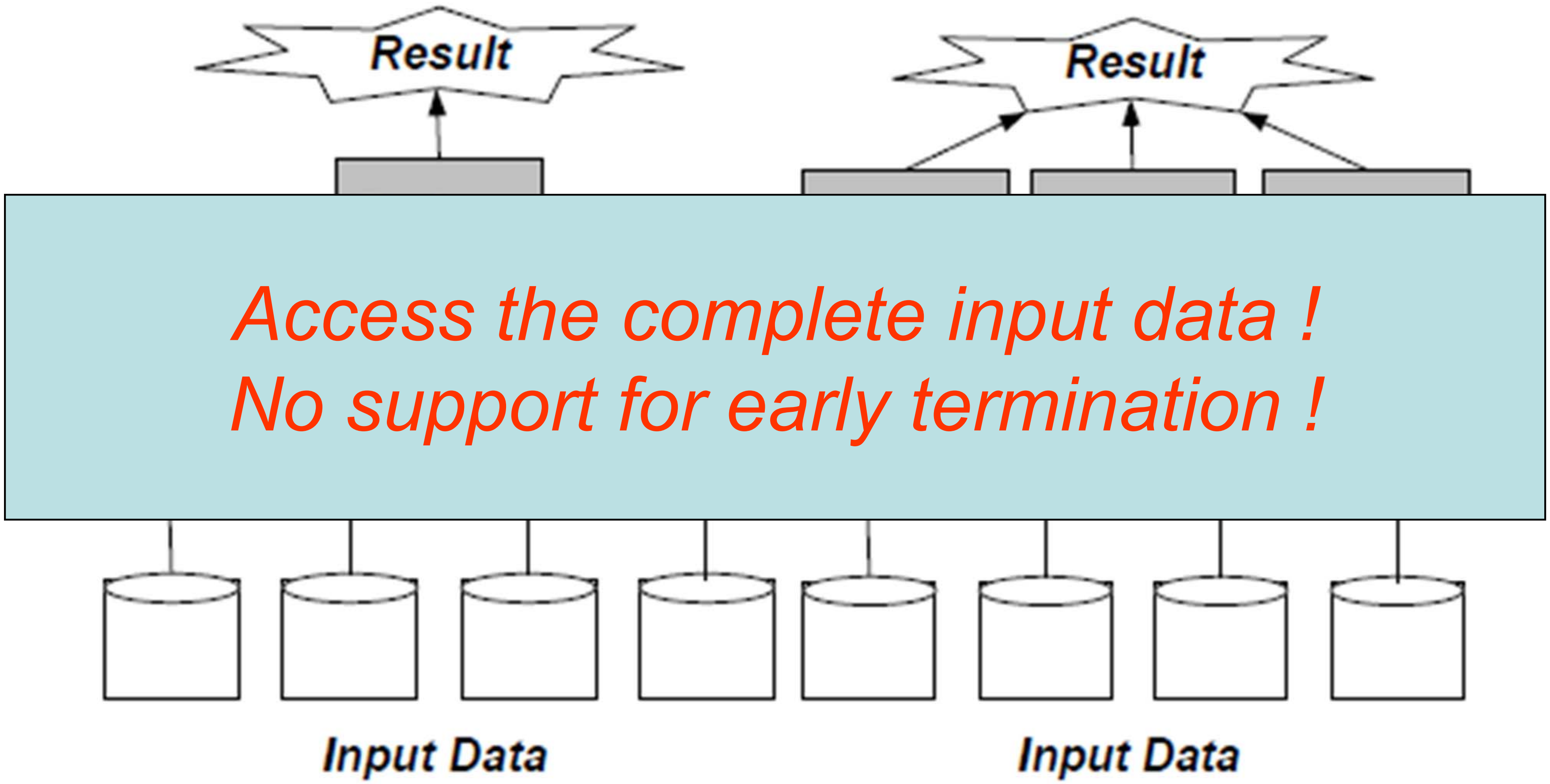
Top-k Joins

ID	NAME	CITY	PRICE
h1	BEST	Trondheim	1400
h2	ABC	London	1200
h3	HOLIDAY	Trondheim	1500
h4	CHEAP	Trondheim	1200
h5	HILTON	London	1800

ID	AIRLINE	TO_CITY	PRICE
f1	KLM	Trondheim	5000
f2	KLM	London	4200
f3	SAS	Trondheim	3000
f4	SAS	Trondheim	3500
f5	KLM	London	2000

ID	NAME	CITY	PRICE	ID	AIRLINE	TO_CITY	PRICE
h1	BEST	Trondheim	1400	f1	KLM	Trondheim	5000
h1	BEST	Trondheim	1400	f3	SAS	Trondheim	3000
h1	BEST	Trondheim	1400	f4	SAS	Trondheim	3500

Rank-aware Processing in MapReduce



(a) Top-k query.

(b) Top-k join.

Rank-aware Processing in MapReduce

Our contributions:

- Sorted access for top-k queries
- Intelligent data placement
- Use of data synopses

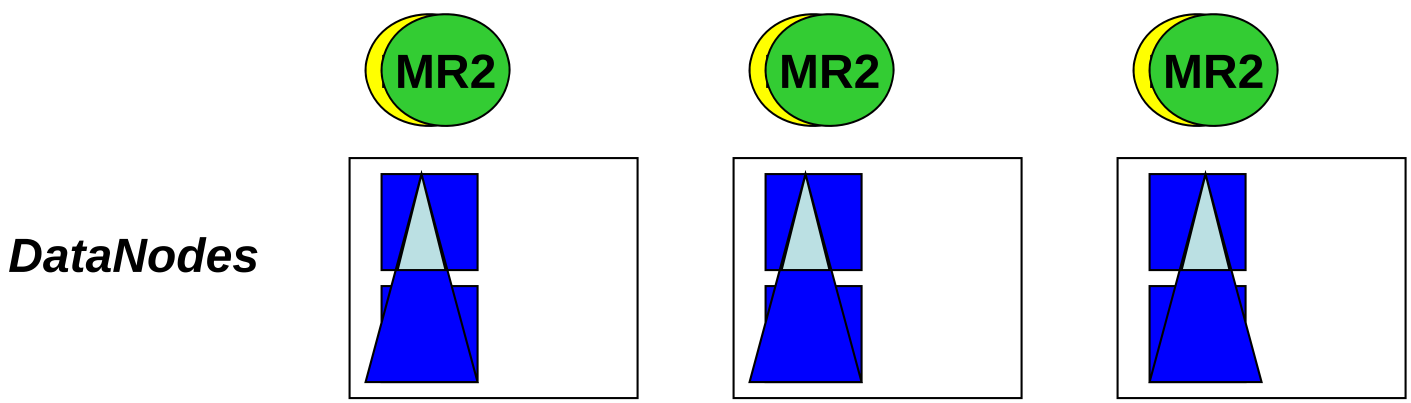
1. Sorted Access for Top-k Queries

- In centralized DBs, efficient processing of top-k queries relies on *sorted access* to data
  - Directly: data is stored sorted on disk
  - Indirectly: provided by a secondary index

- *How can we provide sorted access in MapReduce to support top-k queries $q_k(f)$?*
  - Two alternative techniques
    - Always sort data before top-k processing based on f *(query-dependent sorting)*
    - Store data sorted based on a scoring function $F \neq f$ *(query-independent sorting)*

Query-dependent Sorting

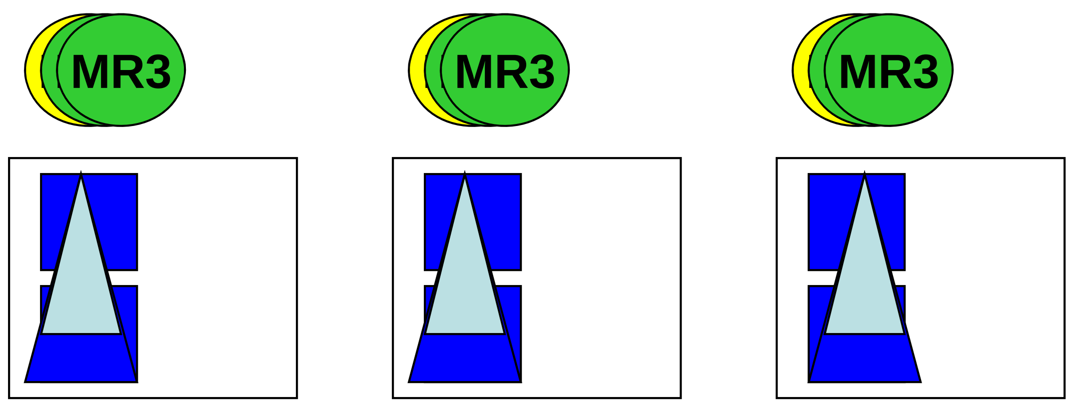
- Use a separate MR job to sort the data based on *f* before processing the top-k query $q_k(f)$
  - Easy to find the top-k results
    - Simply report the k first tuples
  - High overhead to sort data for each incoming query
    - Sorting is query-dependent on scoring function *f*



Query-independent Sorting

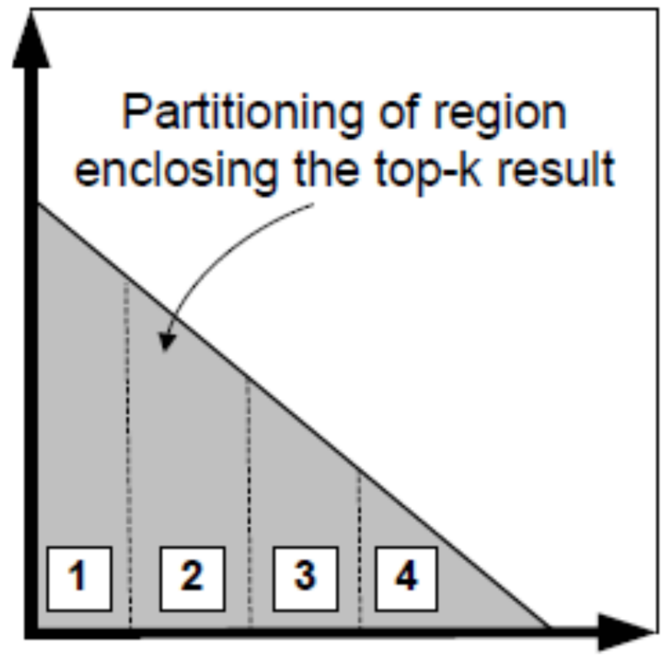
- Store data sorted on DataNodes based on a scoring function *F* (different than *f*)
- To process query $q_k(f)$ it suffices to access only the *K* first tuples of the stored data (where $K > k$)
  - Sorting based on *F* is a one-time cost
    - Again, can be performed using a separate MR job
  - The difference (extra cost) in the number of accessed tuples *K*-*k* increases when *F* differs much from *f*

DataNodes



2. Data Placement

- Data placement on DataNodes affects performance
  - Determined by a *Partitioner* class in Hadoop
- Existing partitioning schemes (e.g., range or hash-based) used for data placement are oblivious to the nature of top-k queries
- Example:
  - *RanKloud [IEEE Multimedia'11]*

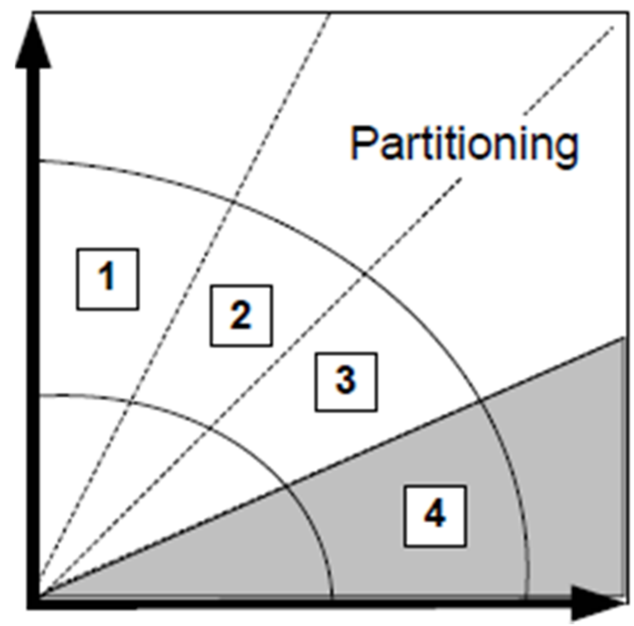


- Desiderata
  - Balance the useful work to DataNodes
  - Avoid redundant processing

Intelligent Data Placement for Top-k Queries

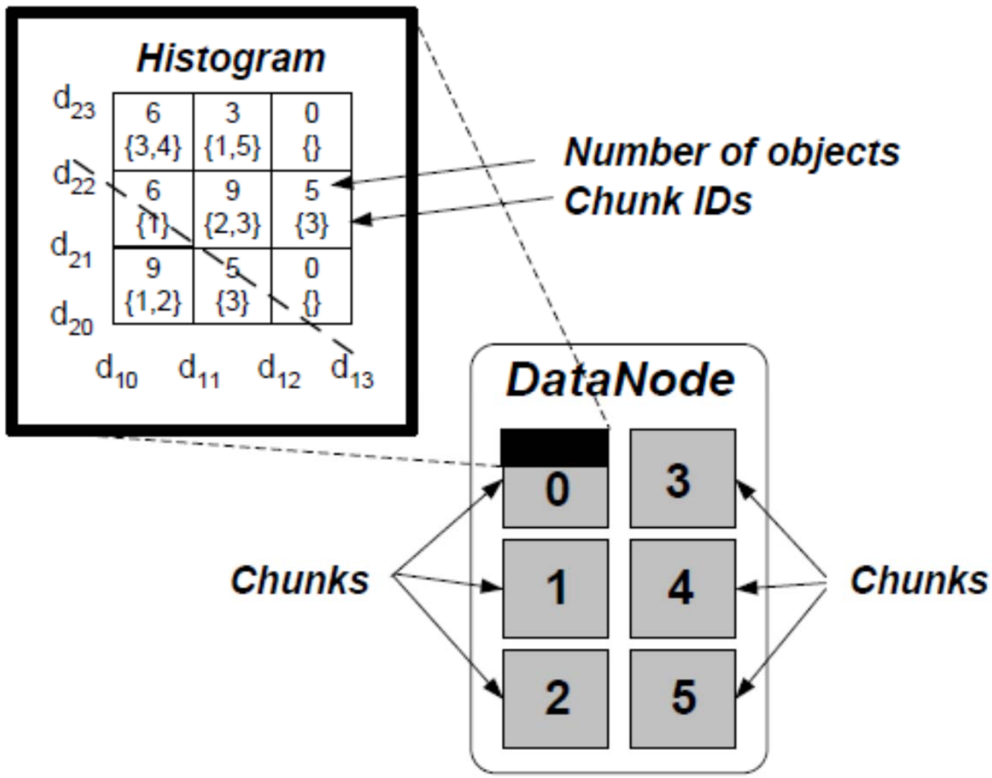
- Angle-based partitioning [SIGMOD'08] (proposed by our group in the context of skyline queries)
  - Splits the useful work fairly
  - Splits the region near the origin of the axes to all partitions

- *Advantages*
  - More intuitive for top-k queries
  - Easy to generalize in higher dimensions
  - Can be combined with sorting
    - Sort based on distance to origin

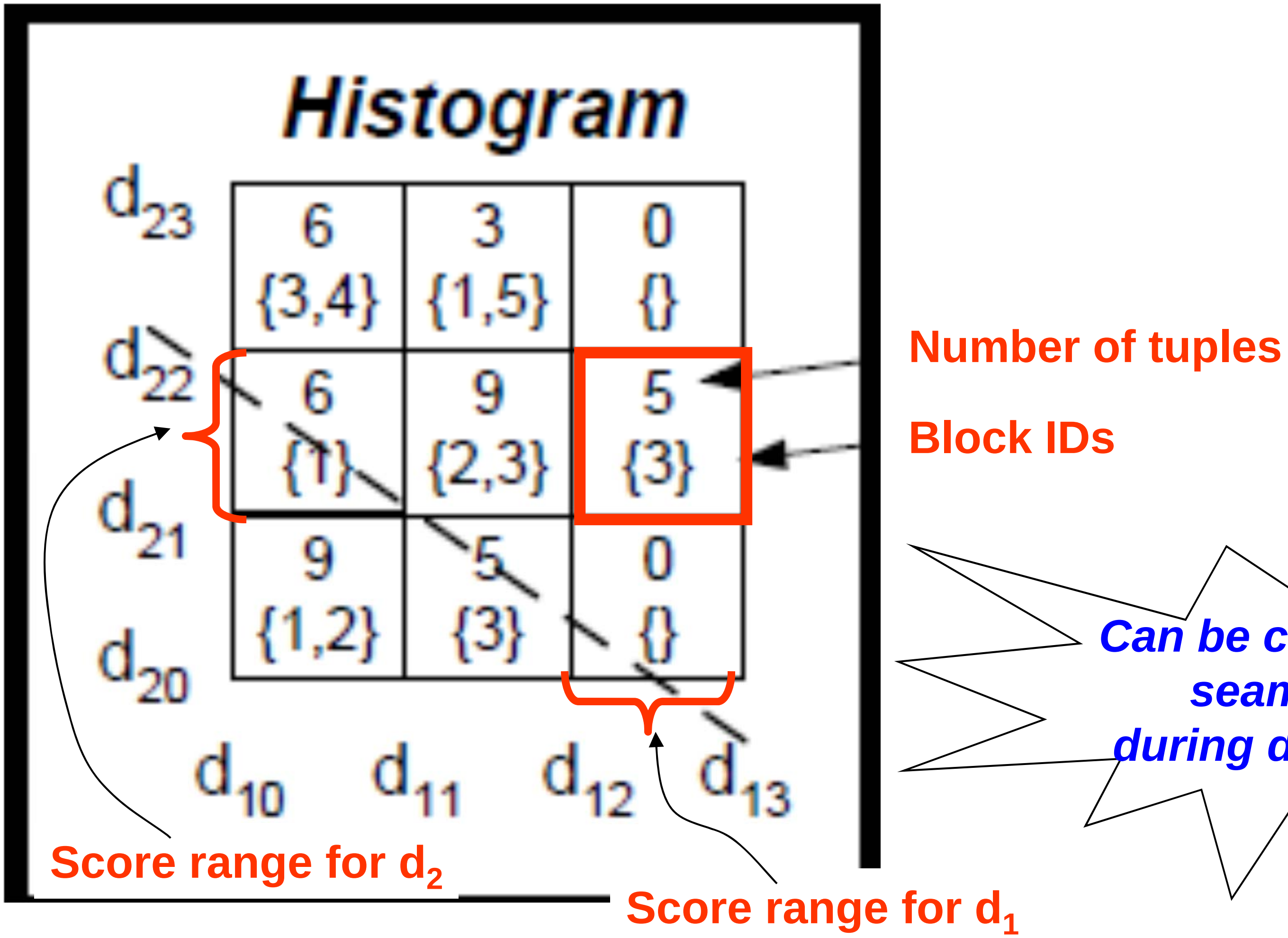


3. Data Synopses

- Main idea
  - Create and store metadata together with data
  - Exploit the metadata during query processing to access only those blocks that may contain top-k results
- Metadata in the form of multidimensional histograms

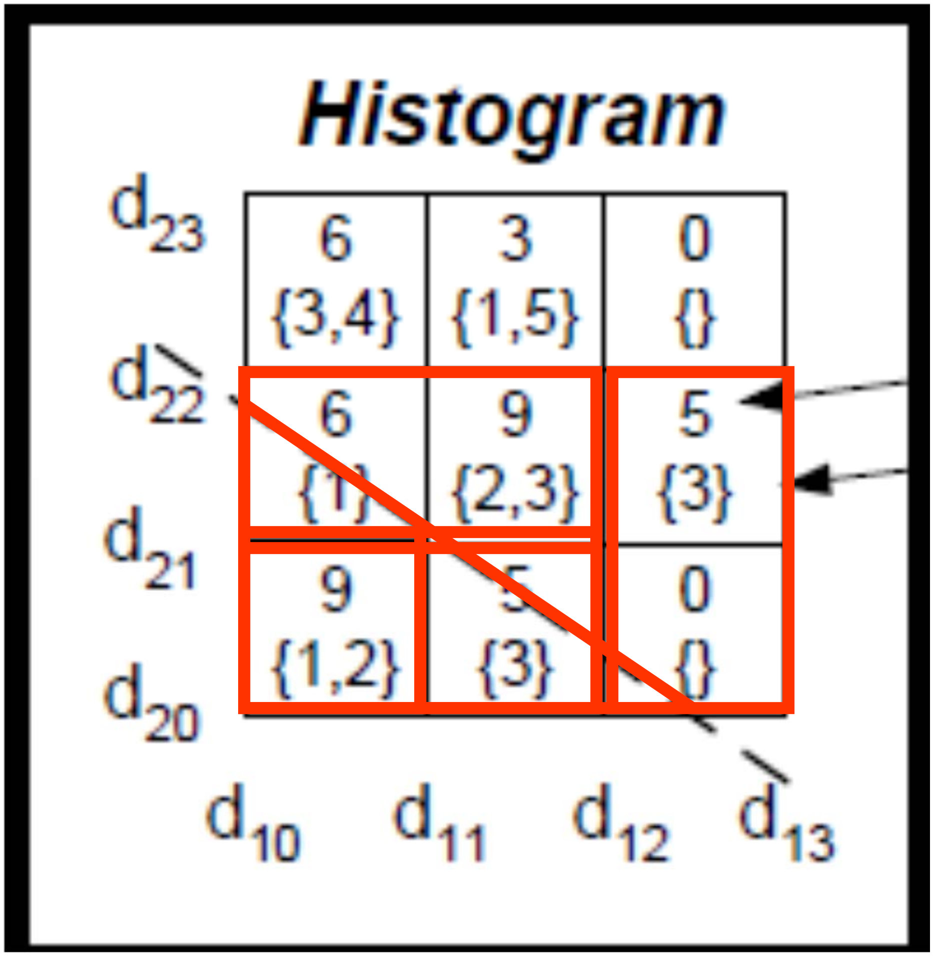


Metadata for Top-k Queries



Can be constructed seamlessly during data upload

Top-k Query Processing Exploiting Data Synopses



- Progressively access histogram bins
  - Until it is guaranteed that the top-k tuples are enclosed in the bins
  - Use upper bound on score
- Retrieve block IDs
- Use random access to retrieve only these blocks

- Example ($k=5$)
  - Access bins up to d_{13} and d_{22}
  - Total of 34 tuples (> 5)
  - Block IDs = {1,2,3}

Optimizations

- Cost model for deciding when the cost of random access of few blocks is smaller than sequential access of many blocks
- Use optimized histograms (e.g., equi-depth)
- Use more advanced methods for histogram bin exploration
  - E.g., examine more bins from the dimension that is more promising to produce the top-k results faster

- *The use of data synopses can be combined with sorting and intentional data placement to boost the performance of query processing*

Related Work on MapReduce

- *RanKloud* [IEEE Multimedia'11]
  - Proposed for top-k join queries
  - Cannot guarantee retrieval of k results
- *CoHadoop* [PVLDB'11]
  - Co-location of files on the same DataNode
  - Useful for joins
- *EARL* [PVLDB'12]
  - Mechanism to stop execution of MapReduce jobs on demand

Conclusions & Outlook

- An overview of techniques for supporting rank-aware processing in MapReduce
  - Sorting, Data placement, Use of data synopses
- Currently, we evaluate these techniques
- Future work
  - Analytical cost models
  - Optimal partitioning scheme for top-k queries
  - More complex query functions
  - Extend the techniques to be applicable for intermediate results produced by other MapReduce jobs

NTNU
Norway

More information:

http://www.idi.ntnu.no/~cdoulk/

cdoulk@idi.ntnu.no