Complex Group-By Processing in XML

to appear in 23rd IEEE ICDE 2007, Istanbul, Turkey.

for **CS631 – ITDBMS**

B. Aditya Prakash (03005030)

Chaitanya G. Gokhale (03005019)

Talk Outline

- Motivation
- Related Work
- Our Contributions
- Query Structure
- Algorithm : Nested-Group-By
  - Dealing with *having*
  - Moving Windows
- Experimental Analysis
- Conclusions & Future Work

Motivation

- Emergence of XML as a popular data-exchange standard
- Group-by queries are one of the most common class of practical queries
- BUT – XQuery
  - Has no explicit group-by operator
  - And requires *simulation* of group-by operations by nesting
- Hence focus on efficient processing of a group-by operator (additionally with aggregation etc.)

Related Work

- Beyer et al [1] and Borkar and Carey [2] propose syntactic extensions to XQuery FLOWR expressions to support explicit for group-by.
- But none discuss algorithms to directly support group-by.
- Another approach: Detect grouping in nested queries and rewrite with explicit grouping operations.

Related Work (contd.)

- Most popular approach: Shred the XML data to tables in Relational database and execute equivalent SQL query [3]
  - Works for fixed schemas. Need to re-shred frequently for dynamic schemas – inefficient
  - Conversion of XQuery to SQL – not automatic
  - Loss of expressive power of XML (hierarchy etc.)
  - Performance issues in nested/hierarchical queries

  More on this later.

Our Contributions

- Framework for expressing complex group-by queries on XML with a variety of aggregation, nesting, having clause etc.
- A disk-based algorithm for efficient processing of the above queries
- Stringent experimental performance analysis

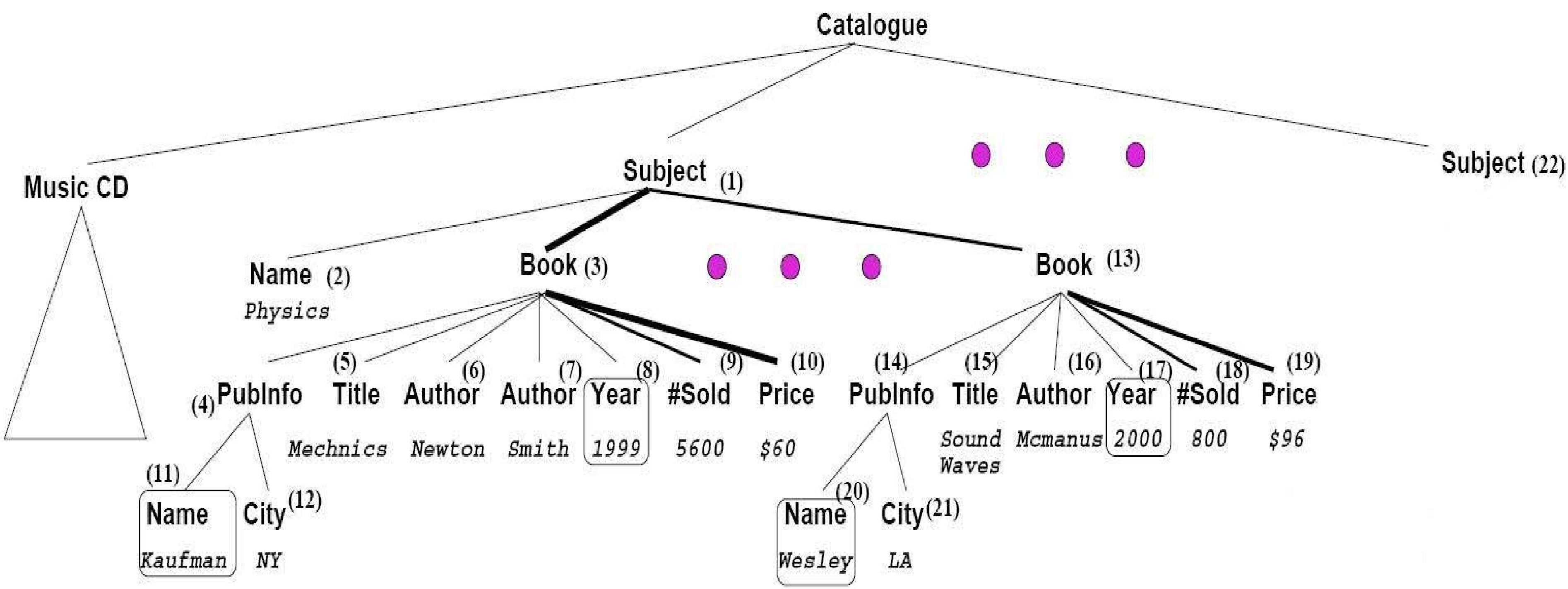
Example Group-by Query

- Consider the following query:

```
group //Book
by //Name return (
     //Name, avg(/Price), count(*)
     then by /Year return (
          /Year, median(/#Sold)
     )    )
```

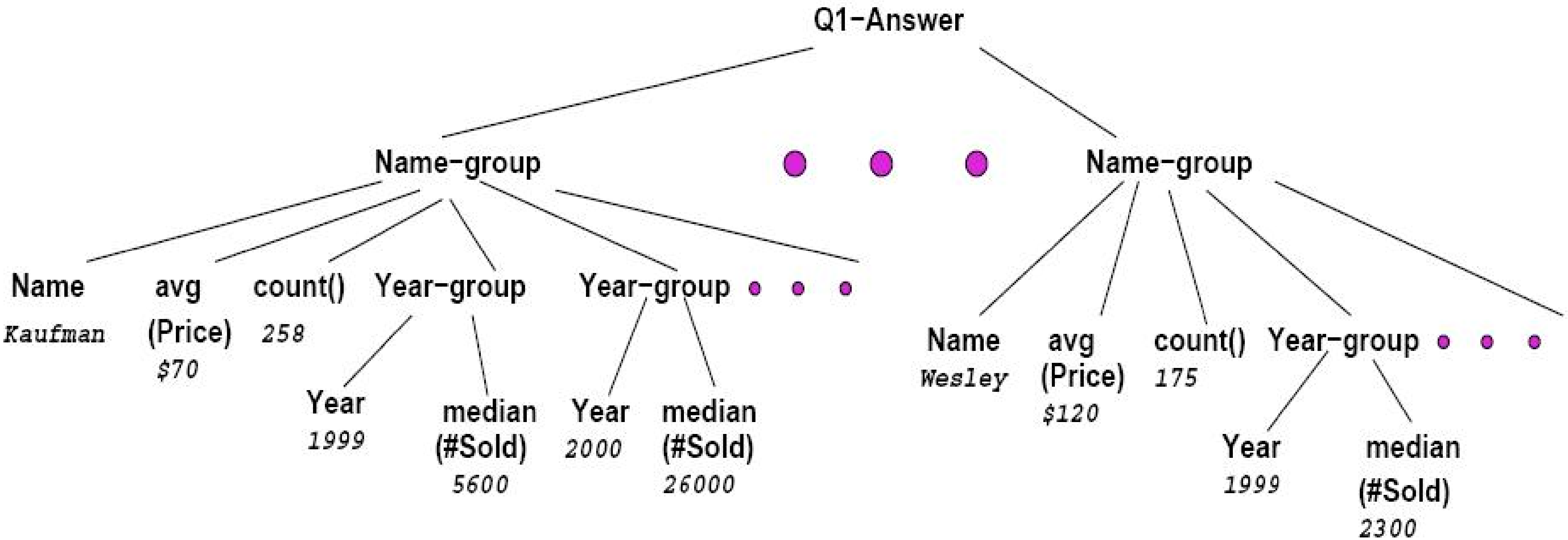
Example (Contd.)

- On this data:



Example (Contd.)

- To get this Answer Tree:



General Query Framework

- General form of a nested query :

$$
\begin{array}{l}
\textbf{group } \alpha \\
\textbf{where } Cons \\
\textbf{by } \beta_1^{out}(mw_1^{out}) \ldots \beta_k^{out}(mw_k^{out}) \\
\textbf{having } AggCons^{out} \textbf{ return } (\\
\quad \beta_1^{out}, \ldots, \beta_k^{out}, agg_1^{out}(\gamma_1^{out}), \ldots, agg_m^{out}(\gamma_m^{out}) \\
\quad \textbf{then by } \beta_1^{in}(mw_1^{in}), \ldots, \beta_p^{in}(mw_p^{in}) \\
\quad \textbf{having } AggCons^{in} \textbf{ return } (\\
\qquad \beta_1^{in}, \ldots, \beta_p^{in}, agg_1^{in}(\gamma_1^{in}), \ldots, agg_q^{in}(\gamma_q^{in}) \\
\quad) \quad)
\end{array}
$$

Framework (Contd.)

- Aggregation Operations : All of *agg*'s are aggregation operations such as min(), count(), sum(), median() etc. applied on γ's.
  - Aggregations can be nested e.g. maxMedian()
- Conditions :
  - *AggCons* are sets of aggregation conditions.
  - *Cons* in the where clause are node-level selection conditions.

Framework (Contd.)

- Moving Windows : *mw*'s denote moving window specifications.
  - *mw* ≡ { width, step, winType, domType }
  - winType = cumulative, fixed etc.
  - domType = active or standard
    - active domain : include only those values that appear in database.
    - standard domain : include all the possible values
  - Percentiles can be expressed too

Algorithm : Nested-Group By (NGB)

- The algorithm can be divided into three steps:
  - Initialization : construction of canonical tree
  - Merge phase : apply node merge operation repeatedly
  - Answer extraction : Aggregate computation

NGB Algorithm (Contd.)

- Initialization
  - Identify nodes of type α, β and γ's, while pruning nodes of other type. Outcome is a canonical tree following input data tree structure.
  - Compute group-by labels from β values and associate them with α nodes.
  - Based on type of aggregate function, associate appropriate quantifier e.g simple counter for count(), frequency table for median() etc.

NGB Algorithm (Contd.)

- Merge Phase :
  - Processing β nodes :
    - if we have a new value create new node in answer tree with appropriate group-by-label,
    - else update the existing node corresponding to this group-by-label.
    - e.g. for Q1 first time Name = Kaufman is encountered, a new Book node is created in the answer tree as a child of root node.

NGB Algorithm (Contd.)

- Processing γ (gamma) nodes :
  - Two cases need to be considered..
  - **Holistic aggregation function** such as median(): All values for the specific β combination need to be collected before aggregation
    - Values accumulated frequency table in main memory
    - In Disk-based version : values written out to a file, called **gamma file.**
  - **Non-Holistic aggregation function** such as sum(), average(): aggregation can be computed on-the-fly by appropriate updates to a suitable finite set of counters.

NGB Algorithm (Contd.)

- Pseudo-code for the algorithm :

```
Algorithm NGB-Disk
Input: XML tree-file, query
Output: answer tree
  (1) Open input file and initialize answer tree.
  (2) for each node encountered {
  (3)   if the node is not an α, β, or γ node, skip the node
  (4)   if it is an α node {
  (5)       update appropriate counter if count(*) is specified
  (6)       if node type inversion is involved, update the dummy α node }
  (7)   if it is a β node {
  (8)       if a new β value is encountered
  (9)           create a new set of group nodes with the group-by label
  (10)      otherwise, update appropriate counters if count(*) is specified }
  (11) if it is a γ node {
  (12)      if the aggregation is holistic,
  (13)          output the value and the α-node id to the gamma file
  (14)      otherwise {
  (15)          if the parent-id associated with the counter is the same
                   as the parent-id of the current node, invoke domergesiblings()
  (16)          otherwise, invoke domergenonsiblings() } }
  (17) } /* end-for */
  (18) scan through the gamma file, using the α-node ids to form groups
  (19) use domergenonsiblings() to compute the aggregation for each group
  (20) put the computed values in the appropriate ndoes of the answer tree }
```

Dealing with *having*

- Naïve solution:
  - Compute the aggregation
  - Then apply the having clause
  - Unnecessary computation!
- Anti-Monotonic Early Pruning
  - A constraint that remains false once it is first violated
  - E.g. *count(*) < 100*, *min(Price) > 100* etc.
  - Convertible constraints [4]
  - Helps in many cases especially during Nesting

Moving Windows

- Repeated Aggregation Strategy (RA)
  - Most natural way
  - For each window we create a answer tree node
  - Each β value hashed to its ‘window’ nodes
    - hash(β) may be NULL if *step > width*
  - Update quantifiers of ALL corresponding ‘window’ nodes whenever we find a β value
  - Aggregation may need to be repeated!
  - But, is better some times!

Moving Windows (Contd.)

- Rolling-Over Strategy (RO)
  - Given query Q, consists of 2 stages
    - Run Q(mw’) – formed by removing the mw specification in Q
    - Outcome is T(mw’)
    - Use T(mw’) to form final answer
    - Specific computation depends on aggregation fn.
      - Non-holistic (distributive and algebraic) functions can be rolled-over from window to window. E.g. sum(), avg()
      - For holistic functions, maintain a frequency table. Now this can be rolled over

Moving Windows (Contd.)

- Example
  - Let range of values be [1991, 2006]
  - Let width = 5, step = 1
  - RA Strategy –
    - Nodes for windows [1991, 1995], [1992, 1996] .. etc
    - So repeatedly aggregate for all windows where say, 1992 is encountered – hence 1991-1995, 1992-1996 are updated
- RO Strategy –
  - Nodes formed ONLY for each of 1991-2006
  - Aggregate each of them only ONCE
  - If sum() is used
    - say, calculation for [1991, 1995] has been done
    - Now roll-over – just subtract Agg(1991) and add Agg(1996)

Moving Windows (Contd.)

- Both are good depending on circumstances
  - When width < step, windows are disjoint
    - RO strategy involves redundant computation
    - Hence RA should perform better
  - When width > step, windows overlap
    - RA strategy does a lot of extra computation
    - Hence RO should perform better
  - Empirical results evaluate the performance trade-off

Moving Windows (Contd.)

- Can be easily extended to handle
  - Nested group-bys
  - Multiple Moving Windows
    - Gives rise to ‘hyper-rectangular’ mws
    - Different from NESTED mws
    - MW on multiple β’s. e.g. year(5, 1), price(10, 5)
  - Combined with *having*
    - Can save a lot of computation if MW is in an inner block and *having* is in the outer

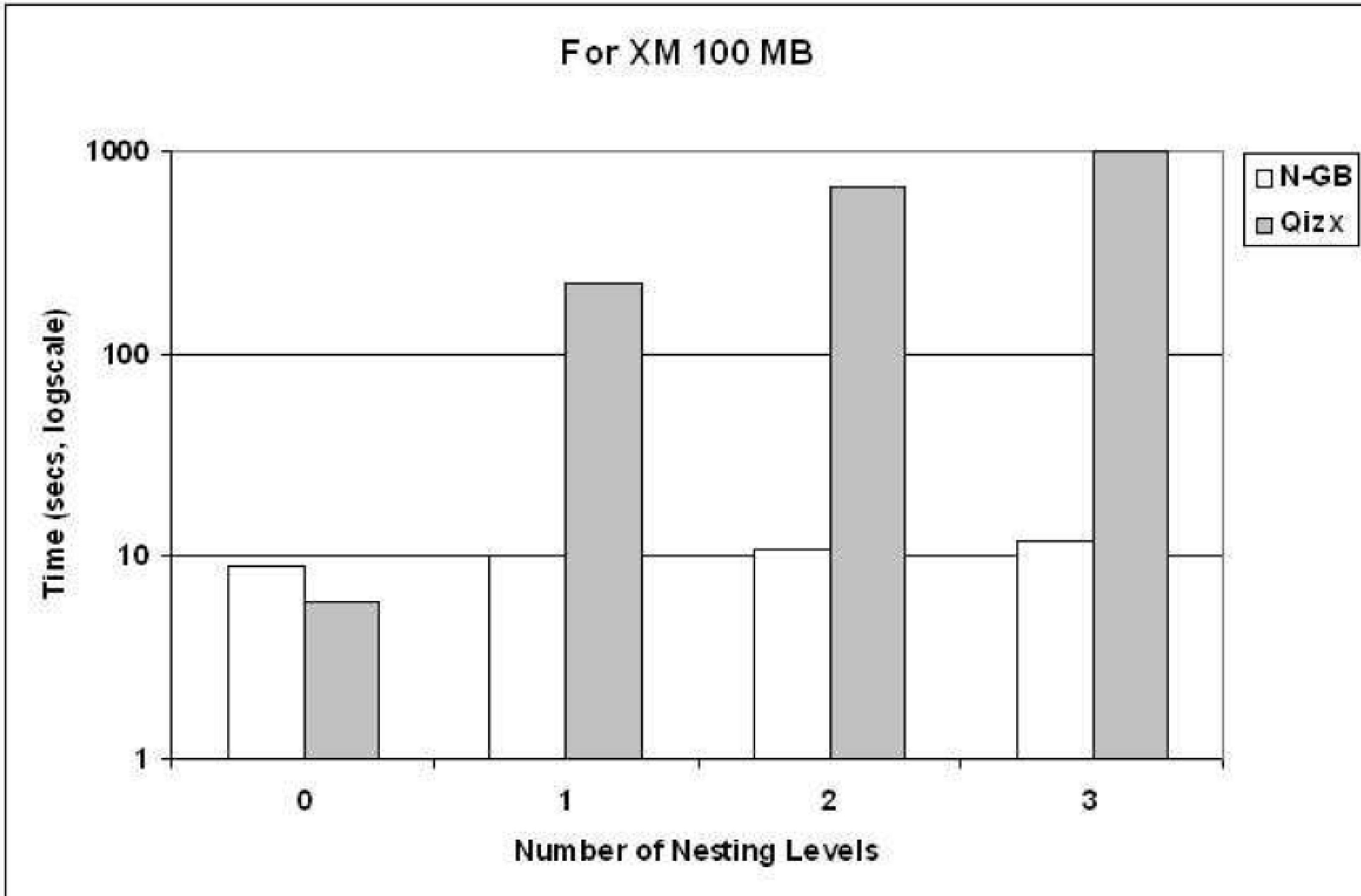
Experimental Analysis

- For comparison, we picked **Galax** – single major complete reference implementation of XQuery, **Qizx** – one of the most efficient XQuery engines available & a RDB(Oracle).
  - Galax performed very poorly taking minutes while we could evaluate in seconds
  - Qizx did well for simple queries but scaled poorly with data size, nesting.
  - Oracle – performed well with flat queries, but degraded with increasing nesting.

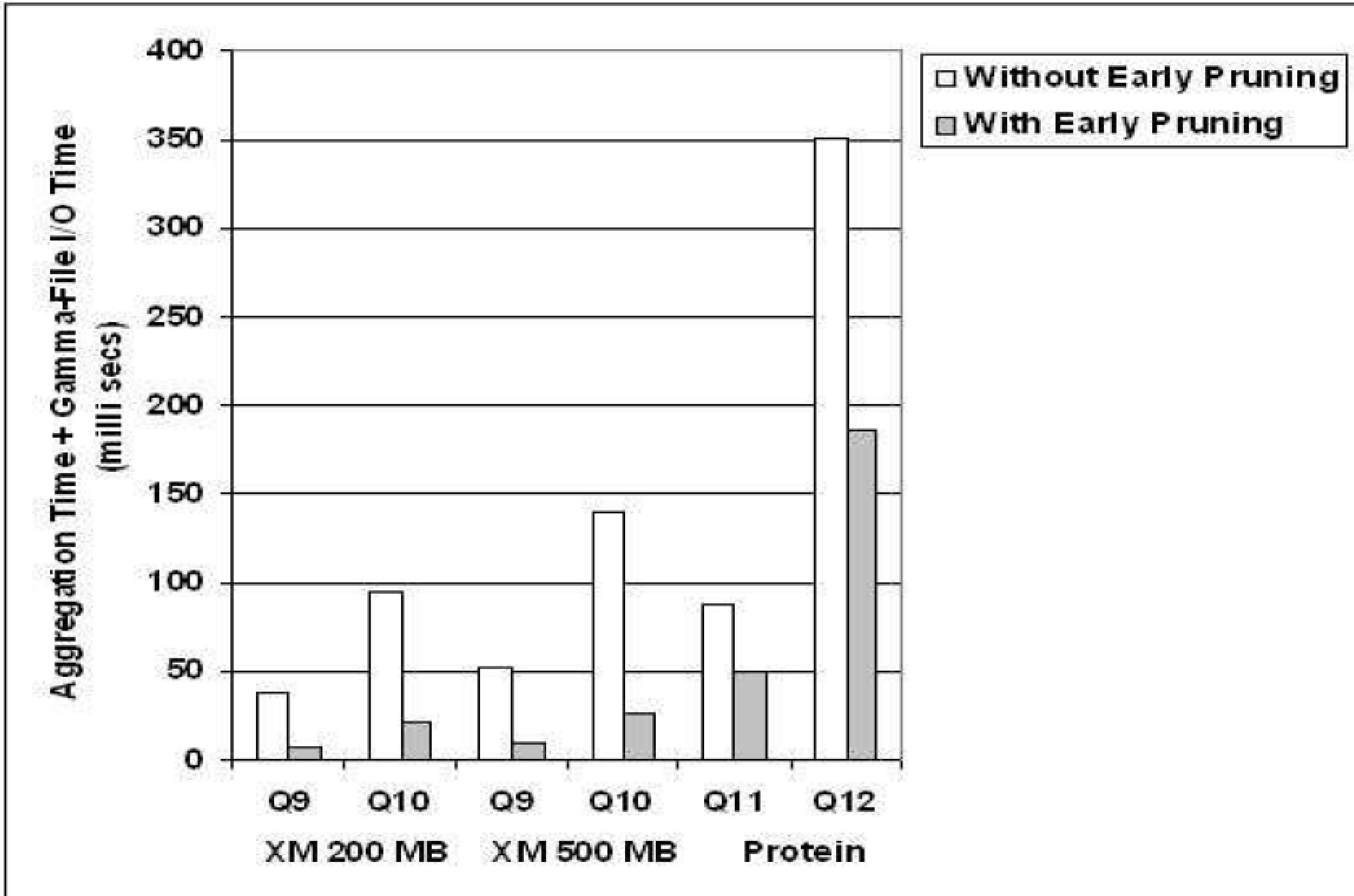
Experimental Analysis (Contd.)

- We also evaluated trade-offs of various strategies discussed in the paper like early pruning, RA vs. RO etc.
- We give graphs for these in the next few slides.

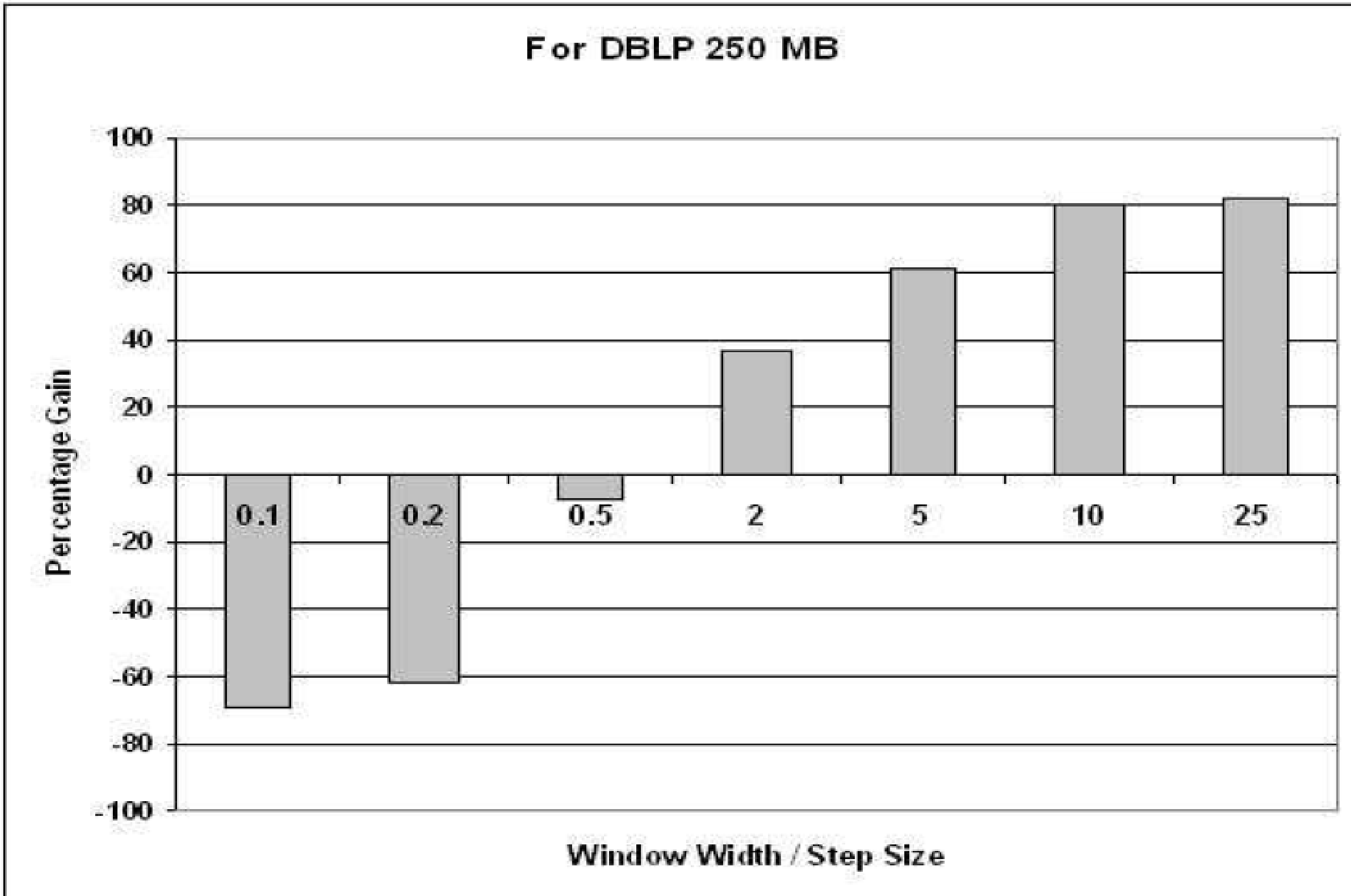
Experimental Analysis (Contd.)



Experimental Analysis (Contd.)



Experimental Analysis (Contd.)



Conclusions & Future Work

- We have an efficient framework for nested group-by queries in XML
- Algorithm NGB has scalability, stability and extensibility
- Challenge now to extend it to data analytics like OLAP etc.

THANK YOU!

- Many details were omitted for brevity. Check out the paper for details.
- This was a joint work with Prof. Laks and Prof. R. Ng at UBC, Vancouver and was supported by Canadian research funds.

- **Any Questions?**

References

[1] K. Beyer et al. “Extending XQuery for Analytics,” SIGMOD 2005, pp. 503–514

[2] V. Borkar and M. Carey. Extending XQuery for Grouping, Duplicate Elimination, and Outer Joins. XML Conference and Expo., Nov. 2004.

[3]J. Shanmugasundaram et al. Relational Databases for Querying XML Documents: Limitations and Opportunities. VLDB 1999: 302-314.

[4] J. Pei et al. Mining Frequent Item Sets with Convertible Constraints. ICDE 2001: 433-442.

Please refer the paper for a more complete bibliography.