COMP5323 Web Database Technologies and Applications

Research Paper Review Report

"Generating Efficient Execution Plans for Vertically Partitioned XML Databases"

Group 7

QING Pei, Edward (11500811g)

LO Wing Yi, Wing (11523479g)

SHAO Shuai, Philip (11552402g)

Table of Contents

I. Paper Information

Title	:	Generating Efficient Execution Plans for Vertically Partitioned XML Databases
Authors	:	Patrick Kling, M. Tamer Ozsu, Khuzaima Daudjee, University of Waterloo Cheriton School of Computer Science
Forum	:	37th International Conference on Very Large Data Bases, VLDB 2011
Download	:	http://vldb.org/pvldb/vol4/p1-kling.pdf

II. Paper Outline

A) Abstract

This paper showed how distributed query evaluation can be performed in a vertically partitioned XML database system. The researcher proposed two distributed optimization techniques and they proved that their proposed techniques could reduce the cost of evaluating local query plans and reduce the input sizes of cross-fragment joins. They used the response time-based cost model to find the best execution plans.

B) Objective

This paper focused on improving the existing problem of generating distributed execution plans within the context of a vertically partitioned and distributed XML database system. The researcher did not concern the executive plans for horizontal and hybrid distribution. They provided several techniques for choosing and optimizing execution plans and presented the improvement of performance and scalability of distributed XML query evaluation. They analyzed the benefits and drawbacks of different types of execution plans, they proposed some optimization techniques to prune irrelevant subtrees in a fragment by pushing crossing fragment joins into local query plans and also compose the costs of local query plans while taking into account parallelism and optimization technique.

C) Background

1) Data Model - Directed Graph Representation

Assumptions:

- Ignores the distinction between XML elements.
- Attributes by treating uniformly as nodes.
- Original schema definition does not contain unspecified portions e.g ANY.
- Extract the information capture by graph representation from DTD/ XML Schema.
- Extracting schema information yields on schema graph which may be less restrictive than the original schema.
- Schema graph is never used for the validation of documents.

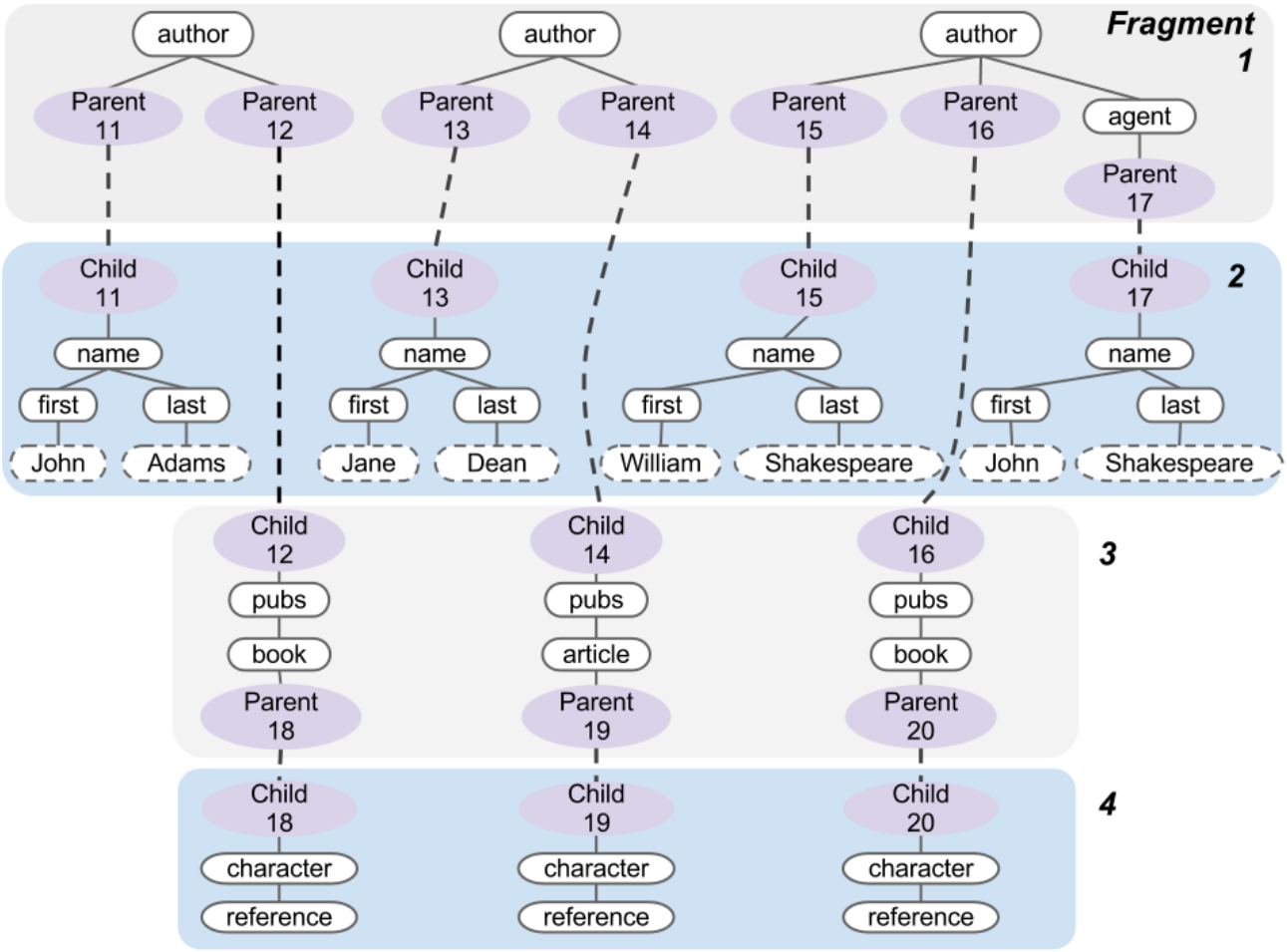
2) Query Model

- Subset of XPath(XQ): XQ consist of absolute location paths consisting of node tests with and without wildcards, child(/) and descendant(//) axes and predicates.
- Query Tree Pattern(QTP) is a tree pattern representation of a query to express queries with multiple extraction points.

3) Schema - Vertical Fragmentation Schema

- Fragmenting the schema graph of the collection into connected sub-graphs.
- The schema graph may contain cycles and then the fragmentation schema be a DAG.

D) Methodology



Example of a vertically partitioned XML collection

1) Vertical partitioning

- To store different types of data at different sites in a distribution system.

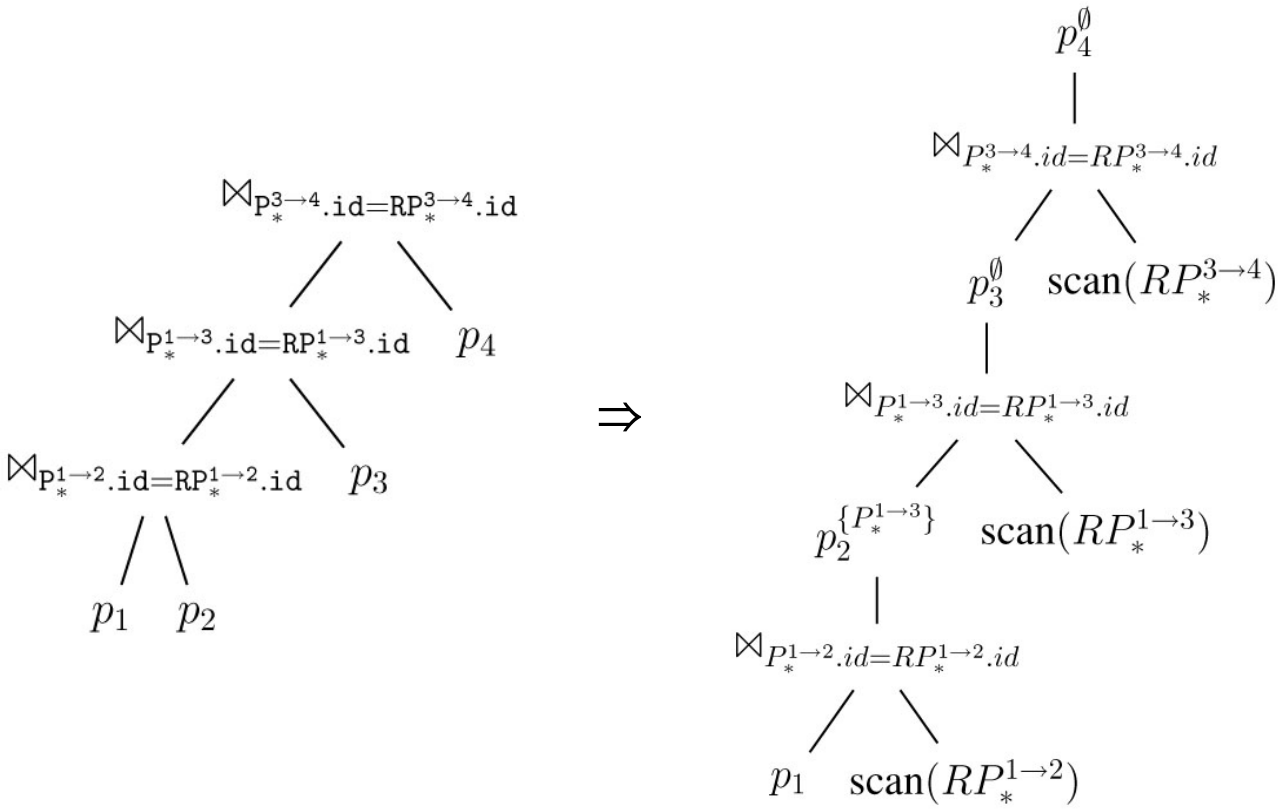
2) Proxy Nodes

- Parent and Child nodes are used as proxies to keep the relation between fragments.

3) Proposed Optimizing Distributed Plans

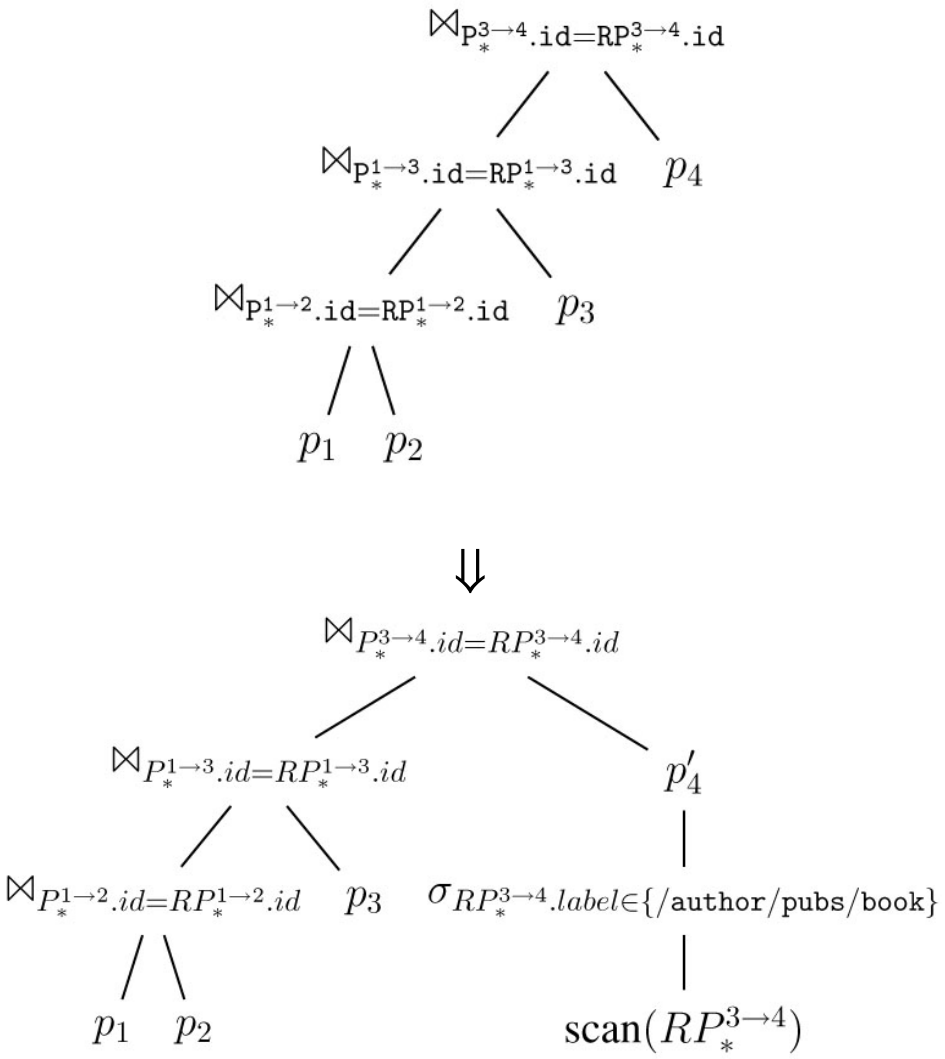
3.1) Pushing Cross-Fragment Joins

The transformation technique from the naive distributed query plan to Pushing Cross-Fragment Joins:



3.2) Label Path Filtering

The transformation technique from the naive distributed query plan to Label Path Filtering:



4) Proposed Cost Model

Response time = Local Execution Time + Joining Time

- Local Execution Time - *snip(i)*: the number of document subtrees accessed by the local plan at *fragment i.*
- Joining Time - *card(i)*: the number of tuples that are returned by the local plan when evaluated at *fragment i.*

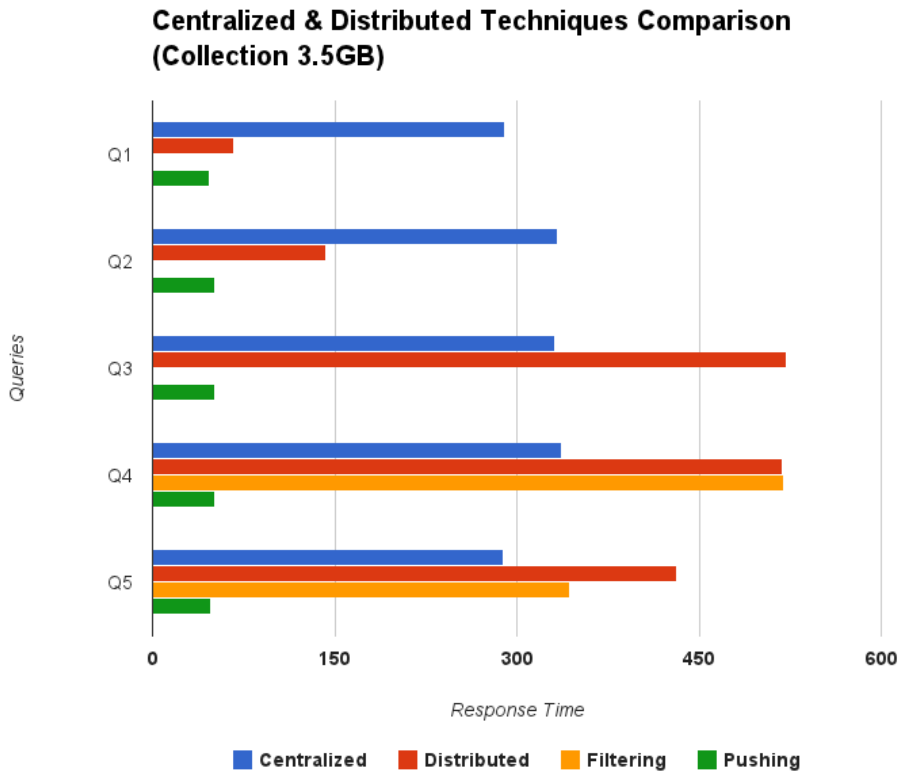
E) Performance Evaluation

1) Resources

- Date Sets: The collections of on-line auction data generated by the XMark benchmark. It is a standard benchmark for evaluating XML query performance.
- Hardware: Virtualized Linux machines within Amazon's Elastic Compute Cloud. A separate instance for each fragment, with an additional instance for dispatching queries. The instance provides 1.7 GB of memory, a single-core 32 bit CPU. All instances run in the same 'availability zone", ensuring low latency, high-throughput communication.

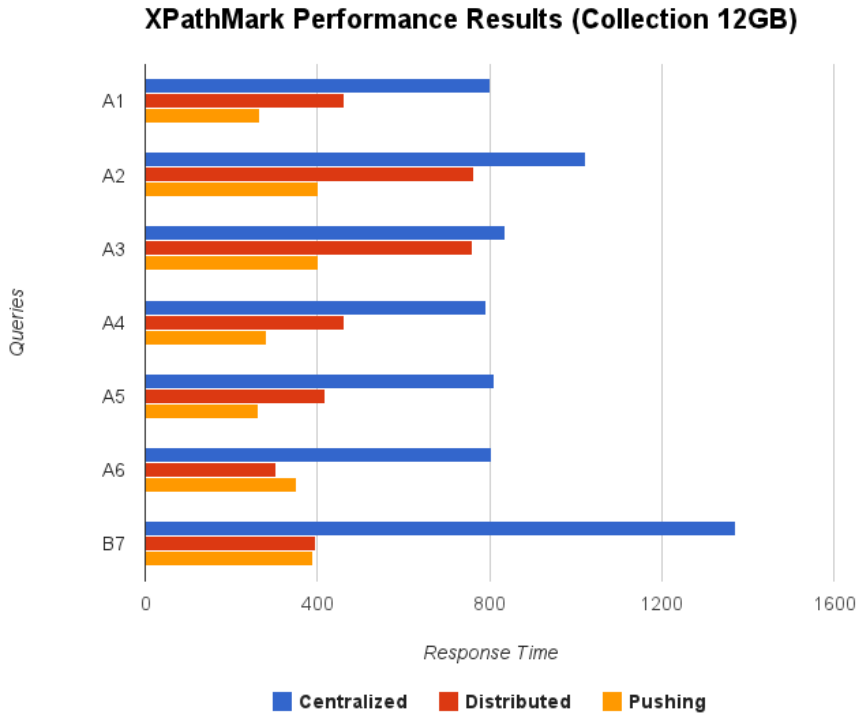
2) Experiment

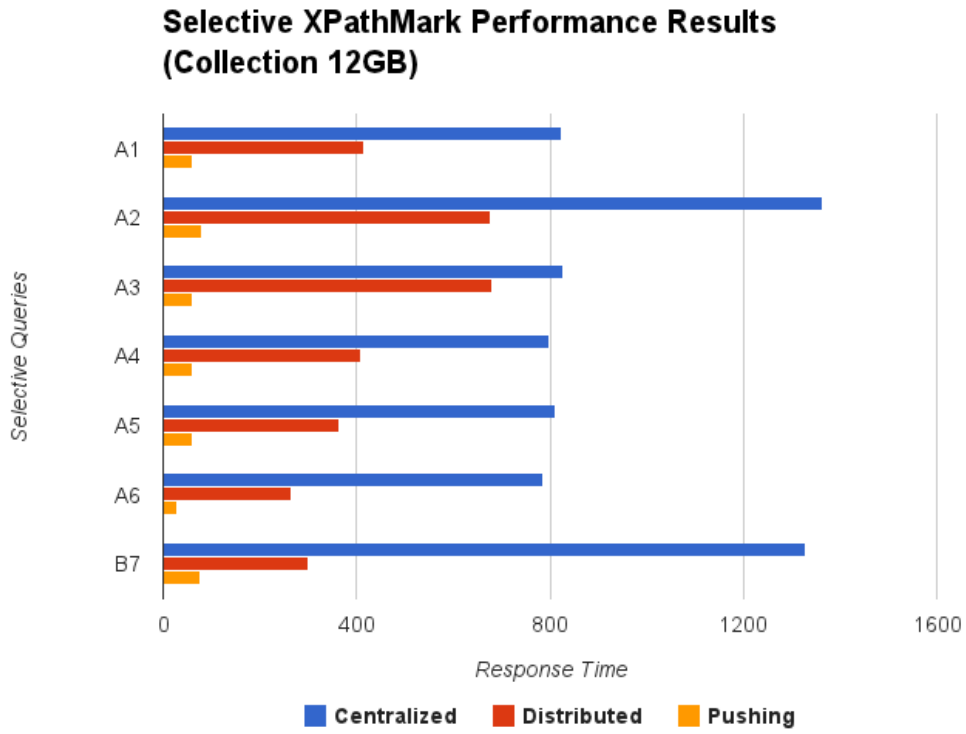
They did two experiments (a) measure the response time achieved by centralized, naive distributed, label path filtering and cross-fragment join pushing query evaluation.



(a) First Experiment Result

(b) measure the response time achieved by centralized, naive distributed and cross-fragment join pushing query evaluation in XPathMark and Selective XPathMark Collection.





(b) Second Experiment Results

F) Conclusion

They proposed a number of plans and optimizations for the distributed processing of queries in vertically partitioned XML database system. The overhead of using our cost model is small, even when exhaustively enumerating all plan alternatives. Choosing the best plan never took more than 0.01 seconds for any of the queries shown. By using their proposed cost estimation model, they found that they proposed two distributed optimization techniques could improve the scalability of XML query processing.

III. Further comments

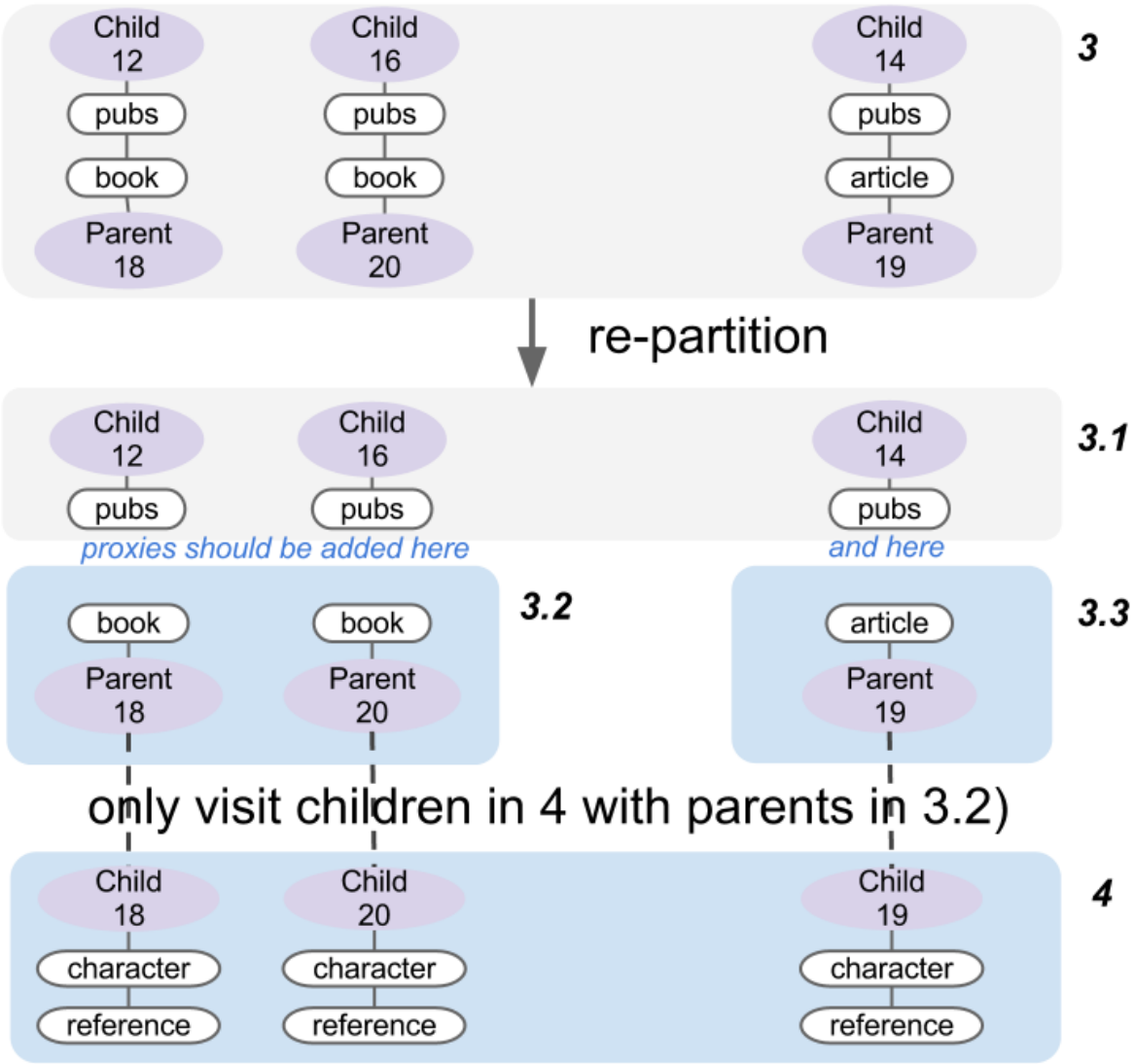
1) Pros and Cons

In this research paper, the researchers used a lot of examples to explain their work step by step, this approach let reader easy to understand and overcome those complicated equations. For example they proposed a cost model to estimate the response time of a distributed execution plan, they wrote a chapter of Cost Model Examples in appendix E to indicate how to calculate the cost of estimation for their proposed optimized techniques - Distributed Execution Plans and Cross-Fragment Join Pushing.

However, they also set many assumptions or limitations on the data model, their proposed distributed optimization techniques may not be applied comprehensively in many real cases.

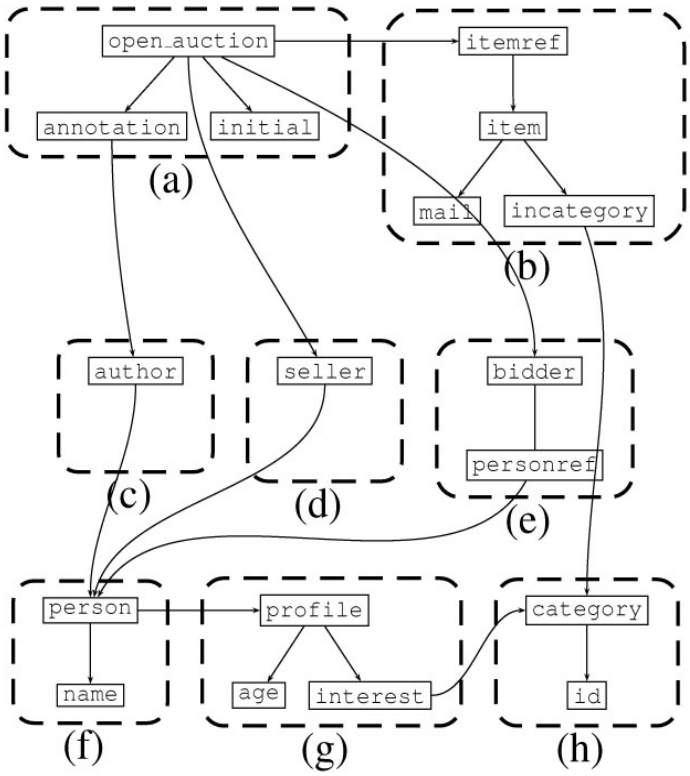
2) Our observation

We have observed that their example of vertically fragmented collection could not present the characteristic of the optimization technique - Pushing Cross-Fragment Joints specifically. Therefore we tried to rearrange the fragmentation and let it more concrete for reader to apply the Push Cross Joints technique. The figure below is rearranged fragmentation:



On the other hand, the response time of the distributed query evaluation with label path filtering (Filt) in the first experiment, they could not obtain the result, it is because the queries Q1-Q3 is not compatible with their fragmentation schema.

- Q1 /open auction[initial > 200]//item//mail/from
- Q2 /open auction[initial > 200][.//author/person/name[starts-with(., 'Ry')]]//item//mail/from
- Q3 /open auction[initial > 200][.//author/person/name[starts-with(., 'Ry')]]//item//category/id



Col.	Q	Response time (s)			
		Cent	Dist	Filt	Push
35MB	Q1	0.85	0.68	N/A	0.70
	Q2	2.74	1.62	N/A	0.90
	Q3	2.71	4.76	N/A	1.00
	Q4	2.85	3.69	3.65	1.07
	Q5	0.72	2.31	1.88	0.90
350MB	Q1	6.90	6.79	N/A	4.96
	Q2	24.78	14.31	N/A	5.50
	Q3	24.73	50.16	N/A	5.63
	Q4	25.43	50.20	50.13	5.66
	Q5	5.71	54.80	42.86	5.14
3.5GB	Q1	289.47	66.93	N/A	47.21
	Q2	332.91	142.90	N/A	51.19
	Q3	331.65	522.10	N/A	51.24
	Q4	336.35	518.92	519.39	51.21
	Q5	288.56	431.42	343.99	47.55

IV. Q&A

Q1. Suggest an advantage and a challenge of vertical partitioning in XML database.

An advantage is the vertical partitioning method could improve the performance of retrieval. It is a challenge to define the distribution model which could suit to the demands of XML, since an XML collection can be partitioned in a largely unconstrained fashion by cutting individual document edges.

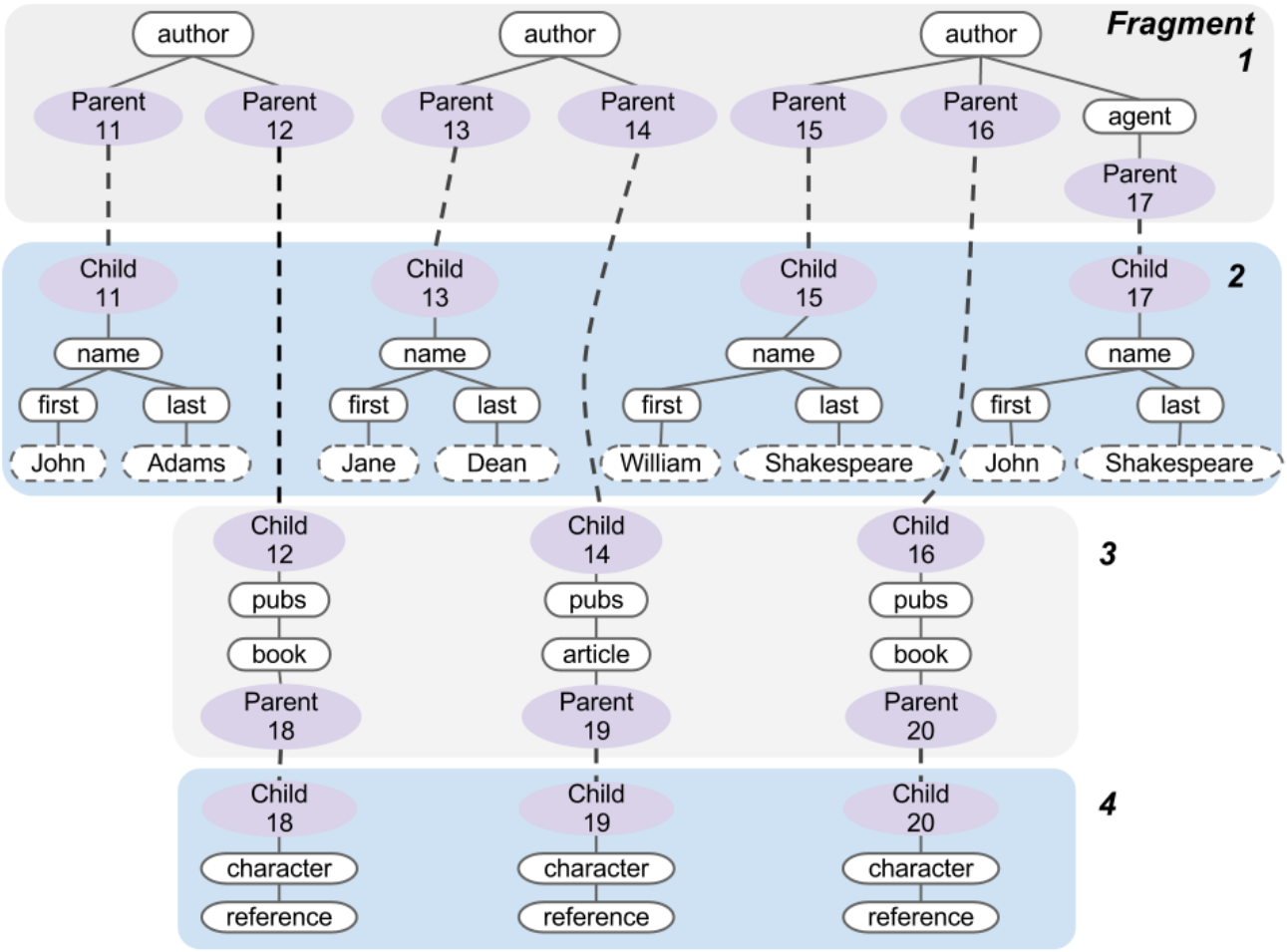
Resource on a single computational node is often limited. Suppose we have an XML collection which is 20GB in size. It is not common to have a single workstation with more than 20GB memory even today. However, having dozens of workstations with 2GB or 4GB memory is much easier. By partitioning the XML collection into fragments of smaller sizes which fit to RAM sizes of querying nodes can greatly increase performance by being away from the low-speed external storage.

Compared to horizontal partitioning, which also cut the XML into pieces, vertical partitioning preserves more parallelism in query execution. Horizontal partitioning partitions the data records while vertical partitioning partitions the tree-structured data model. Vertical partitioning keeps data of the same type or data that are closely connected according to the schema in the same fragment.

Talking about its disadvantage, the increasing number of join operations is a problem when the number of fragments becomes larger. Apart from joins happened in local query execution, results from different fragments have to be joined to form the final output. The more fragments, the more joins. This leads to increasing overhead.

Q2. Are there any replicated data after vertical fragmentation? How can the fragmented XML data in different sites link back together?

No replicated data exist after vertical fragmentation. As shown in the following partition example, each fragment stores data of one or more particular element types. The fragments do not overlap in terms of data coverage.



The method of linking fragments is to add a pair of artificial nodes which are called proxy and root proxy between fragments. For each edge broken during partition, a pair of such nodes are added. Pairs are recognized by a unique ID. In our simplified example, such pairs are denoted as *Parent 11* and *Child 11* and so on. Proxy nodes and root proxy nodes are stored in different fragments as the edge it represents lies across these two fragments.

Q3. How the pushing cross-fragment joins can reduce the size of intermediate results? What is the purpose to enable pipelined execution for join operations?

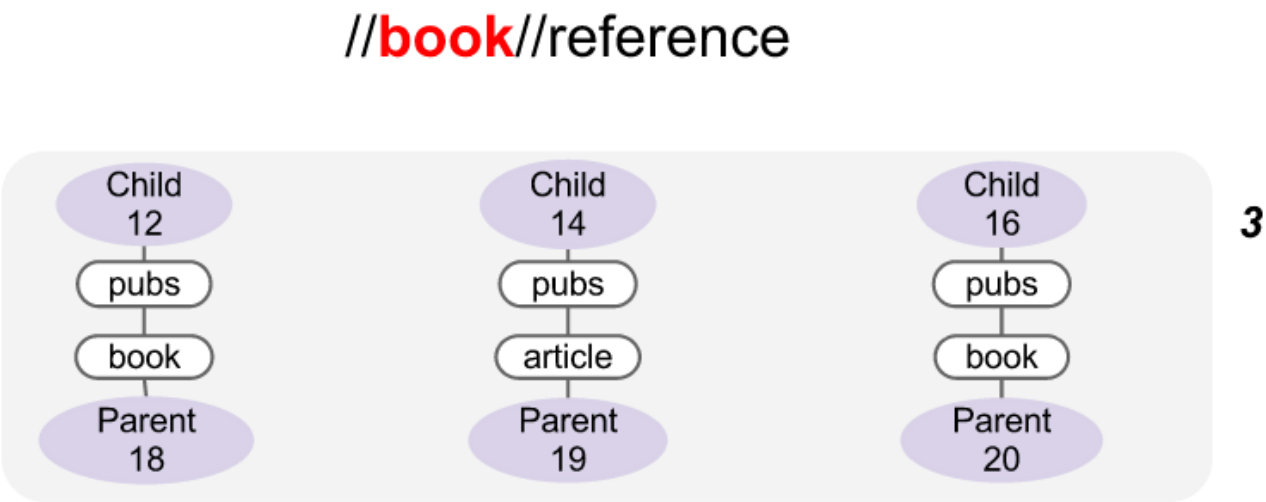
For performance and autonomy reasons, they assume that the sites holding individual fragments are completely independent in how they generate local plans. In order to be able to push cross-fragment joins, they require that the sites holding individual fragments implement. The pushing cross-fragment joins reduces the size of intermediate results that have to be shipped and combined with results from

other sites.

The purpose of enabling the pipelined execution is to greatly reduces the delay introduced by pushing cross-fragment joins. Such that they have to wait only for the first result tuple from the left-hand side of the join before they can start identifying the first relevant subtree.

Q4. Why the label path filtering can help not pushing all cross-fragment joins in a distribution execution plan? The re-writing of a local query plan is done by inserting a selection between the root proxy scan and the remainder plan. Is that the only way of re-writing?

Suppose the user asks for references of books. The query itself has a tree structure. Some nodes in fragment 3 will never match the query tree, i.e. the pubs/article subtree. If we have a mechanism to avoid visiting those subtree for the local query execution, we have a speedup. By adding path labels to root proxy nodes Child 12~14, we make them know the tree structure of its descendants. When the query comes, in the initial scan of root proxy nodes, we immediately remove subtrees that will never match the structure, Child 14 in this case, before trying to match the values if the query specifies any constraints.



The rewriting is done in the current insertion way mainly because the author assumes the generation

of local execution plan is inaccessible to the outside including the node to initiate such rewrites. The only accessible parts are those proxy and root proxy nodes. So everything is done on them.

Another reason is that to reduce the number of nodes to be visited, the filtering is better done as early as possible. The path labels stored in the root proxy nodes have to be read before the process. So it has to be done after the root proxy scan. And that’s it. It can be done later but it would not be optimal to do so.

IV. Appendix

Queries used for evaluation on an XMark Collection

- Q1 /open auction[initial > 200]//item//mail/from
- Q2 /open auction[initial > 200][.//author/person/name[starts-with(., 'Ry')]]//item//mail/from
- Q3 /open auction[initial > 200][.//author/person/name[starts-with(., 'Ry')]]//item//category/id
- Q4 /open auction[initial > 200][.//author/person[profile/age > 30]/name[starts-with(., 'Ry')]]// item//category/id
- Q5 /open auction[initial > 200]//author/person[starts-with(name, 'Ry')]/profile/interest/category/ description

Queries used for XPathMark

- A1 /site/closed auctions/closed auction/annotation/description/text/keyword
- A2 //closed auction//keyword
- A3 /site/closed auctions/closed auction//keyword
- A4 /site/closed auctions/closed auction[annotation/description/text/keyword]/date
- A5 /site/closed auctions/closed auction[descendant::keyword]/date
- A6 /site/people/person[profile/gender and profile/age]/name
- B7 //person[profile/@income]/name

Queries used for Selective XPathMark

- A1S /site/closed auctions/closed auction[price > 600]/annotation/description/text/keyword
- A2S //closed auction[price > 600]//keyword
- A3S /site/closed auctions/closed auction[price > 600]//keyword
- A4S /site/closed auctions/closed auction[price > 600][annotation/description/text/keyword]/date
- A5S /site/closed auctions/closed auction[price > 600][descendant::keyword]/date
- A6S /site/people/person[starts-with(name, 'Ry')][profile/gender and profile/age]/name
- B7S //person[starts-with(name, 'Ry')][profile/@income]/name