Scalable Processing of Flexible Graph Pattern Queries on the Cloud

Padmashree Ravindra
Department of Computer Science
North Carolina State University, Raleigh, USA
pravind2@ncsu.edu

Kemafor Anyanwu
Department of Computer Science
North Carolina State University, Raleigh, USA
kogan@ncsu.edu

ABSTRACT

Flexible exploration of large RDF datasets with unknown relationships can be enabled using ‘unbound-property’ graph pattern queries. Relational-style processing of such queries using normalized relations, results in redundant information in intermediate results due to the repetition of adjoining bound (fixed) properties. Such redundancy negatively impacts the disk I/O, network transfer costs, and the required disk space while processing RDF query workloads on MapReduce-based systems. This work proposes *packing* and lazy *unpacking* strategies to minimize the redundancy in intermediate results while processing unbound-property queries. In addition to keeping the results compact, this work evaluates RDF queries using the *Nested TripleGroup Data Model and Algebra* (NTGA) that enables shorter MapReduce execution workflows. Experimental results demonstrate the benefit of this work over RDF query processing using relational-style systems such as Apache Pig and Hive.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query processing*

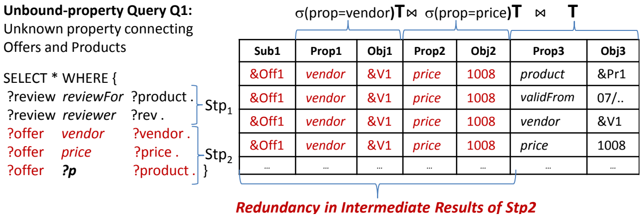
Keywords

MapReduce; RDF Graph Pattern Matching; Unbound-property

1. INTRODUCTION

MapReduce [3] based parallel data processing platforms such as Hadoop [4], Hive [5], and Pig [7] are being leveraged across enterprises to analyse, visualize, and gain insight into high volumes of (semi) structured data produced by data-intensive applications. Exploration of such large scale datasets often requires support for flexible querying based on unknown relationships. In the context of Semantic Web data, *unbound-property* triple patterns can be used to query unknown relationships (“*Scientists related to the same city*”), partially known relationships (“*Proteins related via some kind of interaction*”), or ‘don’t care’ relationships (“*Anything related to Bacteria A*”) that may be useful in data-integration scenarios.

Relational-style MapReduce platforms such as Hive and Pig allow users to express data processing tasks using high-level query primitives, which are translated into logical plan, physical plan, and a sequence of MapReduce (MR) cycles (an MR execution workflow). Complex tasks such as processing RDF graph pattern queries typically involve a sequence of joins over thin relations to reassemble related triples. Such join-intensive workloads result in long execution workflows with multiple phases of I/O materialization, sorting, and data transfer costs. Additionally, the output of each MR cycle is written onto the Hadoop Distributed File System (HDFS) and read back in the subsequent cycle. This overhead is significant in the case of relational-style processing of unbound-property graph pattern queries using normalized relations.



Unbound-property Query Q1:
Unknown property connecting
Offers and Products

```
SELECT * WHERE {
 ?review  reviewFor  ?product .
 ?review  reviewer   ?rev .
 ?offer   vendor     ?vendor .
 ?offer   price      ?price .
 ?offer   ?p         ?product . }
```

Sub1	Prop1	Obj1	Prop2	Obj2	Prop3	Obj3
&Off1	*vendor*	&V1	*price*	1008	*product*	&Pr1
&Off1	*vendor*	&V1	*price*	1008	*validFrom*	07/..
&Off1	*vendor*	&V1	*price*	1008	*vendor*	&V1
&Off1	*vendor*	&V1	*price*	1008	*price*	1008
…	…	…	…	…	…	…

Figure 1: (a) Example unbound-property graph pattern query (b) Star-join result of Stp_2 containing redundant information

$Q1$ in Fig. 1 is an unbound-property graph pattern query with two star subpatterns Stp_1 and Stp_2 corresponding to a Review and a product Offer respectively. The subpattern Stp_2 contains an unbound-property triple pattern ($?offer$, $?p$, $?product$) that specifies an unknown relationship between offer and a product. Stp_2 can be evaluated over a triple relation T using a set of relational joins ($T_{vendor} \bowtie T_{price} \bowtie T$), where T_{vendor} (T_{price}) represents the subset of triples in T with the property type $vendor$ ($price$). The join result of Stp_2 contains redundant information related to the bound properties $vendor$ and $price$, for each triple that matches the unbound-property triple pattern. The redundancy factor in the intermediate results is proportional to the arity and size of the bound-property component and the cardinality of the join involving the unbound-property. Such redundancy negatively impacts the disk I/O, network transfer costs, and the total disk space required for successful completion of a MapReduce data processing task. This work proposes strategies to enable efficient management of intermediate results while processing unbound-property graph pattern queries on MapReduce. The strategies compliment a previous effort using the *Nested TripleGroup Data Model and Algebra* [8, 6, 9] (NTGA) that reduces the I/O footprint of RDF query workloads.

2. APPROACH

NTGA exploits the fact that subgraphs matching ALL star subpatterns in a query can be computed using a single MR cycle by a `GROUP BY` operation on the subject column of the triple relation. This eliminates the need for multiple MR cycles (one for each star subpattern) that are required to evaluate multiple star subpatterns using relational-style joins. A query with n star subpatterns requires n MR cycles using NTGA as opposed to ($2n$-1) cycles using the relational approach. The grouping-based approach results in ‘groups of triples’ or *TripleGroups* that are ‘content-equivalent’($\cong$)

WWW 2013 Companion, May 13–17, 2013, Rio de Janeiro, Brazil.
ACM 978-1-4503-2038-2/13/05.

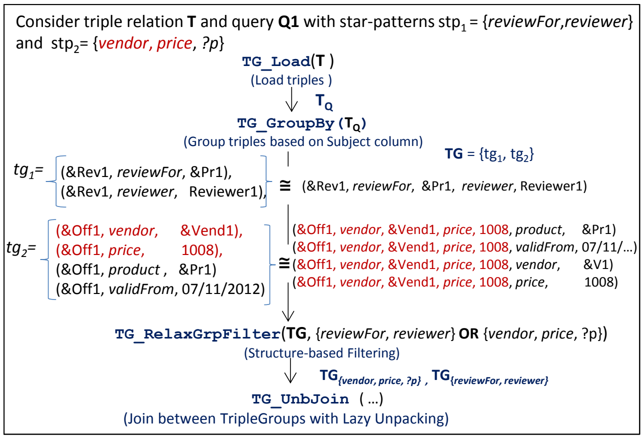


Figure 2: NTGA-based processing of an unbound-property query $Q1$ over a triple relation T

to the n-tuples that result from relational joins. This grouping phase is followed by a filtering step to retain triplegroups that satisfy the join structures (the set of property types) for at least one of the star subpatterns in the query. NTGA operators relevant to this discussion include `TG_GroupBy` (groups the triple relation on the Subject column), `TG_GroupFilter` (retains triplegroups that satisfy the specified structural constraints), and `TG_Join` (joins triplegroups).

For unbound-property queries, NTGA produces compactly '*packed*' triplegroups that implicitly represent the bound-property pattern combinations with each triple matching the unbound-property triple pattern. For example, triplegroup tg_2 in Fig.2 implicitly represents 4 n-tuples that form the star-join result of Stp_2. However, structure-based filtering using the `TG_GroupFilter` assumes a set of bound properties, and needs to be 'relaxed' to allow valid matches to unbound-property star subpatterns. We introduced special operators, (i) `TG_RelaxGrpFilter` to retain triplegroups with a non-empty subset of triples that match the bound properties in a star subpattern (may contain triples with other property types). For example, triplegroup tg_2 in Fig.2 contains matches to the set of bound properties {vendor, price} and is a valid match, and (ii) `unpack` to extract subsets of triples in a triplegroup that match the different pattern combinations corresponding to the unbound-property star subpattern. Triplegroup tg_2 in our example is unpacked into 4 perfect triplegroups (each $\cong$ to one of the the 4 n-tuples produced using the relational join). In order to minimize the redundancy factor in intermediate results, we *lazily* unpack the triplegroups only when absolutely necessary. We propose *lazy map-side* unpacking and *lazy map-side partial* unpacking strategies that unpack the triplegroups only in the MR cycle that processes the join on the unbound-property pattern (using the `TG_UnbJoin` operator).

3. EXPERIMENTAL EVALUATION

The proposed packing and unpacking strategies were integrated into *RAPID*+(NTGA-based extension of Apache Pig) [6]. The performance of RAPID+ was compared with two relational-style MapReduce systems - Apache Pig and Hive, both with tuple-based algebra. Experiments were conducted on NCSU's VCL[1] (each cluster node with dual core Intel X86 machine, 2.33 GHz processor speed, 4G memory) using Pig release 0.10.0, Hive 0.8.1 and Hadoop 0.20.2. Synthetic benchmark BSBM [2] and real-world DBPedia Infobox [1] datasets were used for evaluation. This section provides a subset of the evaluation results. Additional details about the evaluated queries can be found on the project website[2].

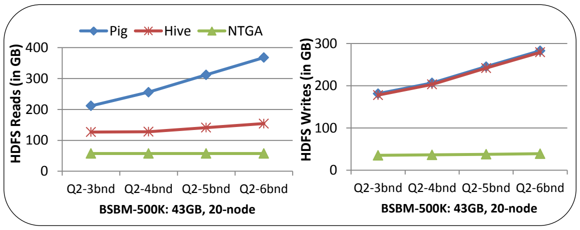


Figure 3: A comparison of HDFS reads and HDFS writes with varying size of bound-property component in unbound-property queries using relational and NTGA-based processing

Fig. 3 presents the results for BSBM-500K (43GB: 500K Products, ≈ 175M triples) on a 20-node Hadoop cluster. Test queries involved two star subpatterns with 1 unbound-property and number of bound-property triple patterns varying from 3 ($Q2\_3bnd$) to 6 ($Q2\_6bnd$) respectively. In general, the increase in the number of bound-property components results in a gradual increase in the size of reduce output for Pig and Hive, since they produce all possible combinations of the bound-component with each triple that matches the unbound-property pattern. The relational approaches produce 10 such combinations for the test queries (relational arity of the subgraph matching the unbound-property subpattern is 10), impacting the total HDFS reads / writes, and overall performance. However, the NTGA approaches compactly capture all the required combinations, resulting in approx. 80 to 86% less HDFS writes than both Hive and Pig for the test queries.

4. CONCLUSION

We presented an approach to *pack* and lazily *unpack* intermediate results of unbound-property graph pattern queries to minimize the redundancy in intermediate results while processing MapReduce execution workflows. Experimental evaluation confirms that the proposed strategies reduce the I/O and network footprint, which can allow more flexible exploration of very large datasets.

Acknowledgements: This work was partially funded by NSF grants IIS-0915865 and IIS-1218277.

5. REFERENCES

[1] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives. Dbpedia: A nucleus for a web of open data. In *ISWC/ASWC*, pages 722–735, 2007.
[2] C. Bizer and A. Schultz. The Berlin SPARQL Benchmark. *IJSWIS*, 5(2):1–24, 2009.
[3] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Comm. ACM*, pages 107–113, 2008.
[4] Apache Hadoop. http://hadoop.apache.org/.
[5] Apache Hive. http://hive.apache.org/.
[6] H. Kim, P. Ravindra, and K. Anyanwu. From SPARQL to MapReduce: The Journey Using a Nested TripleGroup Algebra. *VLDB*, 4(12), 2011.
[7] Apache Pig. http://pig.apache.org/.
[8] P. Ravindra, H. Kim, and K. Anyanwu. An intermediate algebra for optimizing rdf graph pattern matching on mapreduce. *The Semantic Web: Research and Applications*, pages 46–61, 2011.
[9] P. Ravindra, H. Kim, and K. Anyanwu. To nest or not to nest, when and how much: Representing intermediate results of graph pattern queries in mapreduce based processing. In *SWIM*, pages 5:1–5:8, 2012.

---

[1] `http://vcl.ncsu.edu/`

[2] `http://research.csc.ncsu.edu/coul/RAPID/WWW2013`