A Vision for SPARQL Multi-Query Optimization on MapReduce

Kemafor Anyanwu

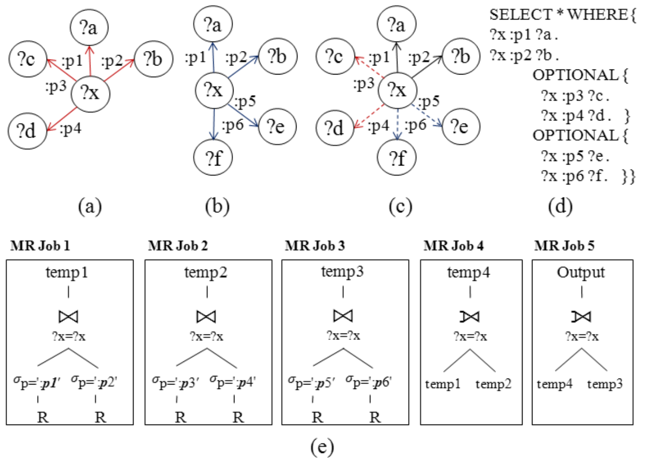
Department of Computer Science, North Carolina State University
Raleigh, NC, USA
{kogan}@ncsu.edu

***Abstract*—MapReduce has emerged as a key component of large scale data analysis in the cloud. However, it presents challenges for SPARQL query processing because of the absence of traditional join optimization machinery like statistics, indexes and techniques for translation of join-intensive workloads to efficient MapReduce workflows. Further, MapReduce is primarily a batch processing paradigm. Therefore, it is plausible that many workloads will include a batch of queries or new queries could be generated from given queries e.g. due to query rewriting of inferencing queries. Consequently, the issue of multi-query optimization deserves some focus and this paper lays out a vision for rule-based multi-query optimization based on a recently proposed data model and algebra, *Nested TripleGroup Data Model and Algebra*, for efficient SPARQL query processing on MapReduce.**

I. Introduction and Problem Statement

MapReduce [1] is emerging as a dominant computational paradigm for large scale analytic-style processing. It enables easy program implementation via a simple programming interface, automatic parallelization of programs and on-demand scaling up using cloud resources. Analytical tasks often involve a batch of queries to be executed and it is not unusual that some queries share common subexpressions. Another potential source of query groups are the recent proposals [2] for optimizing inferencing queries using database-style techniques that rewrite queries into a disjunction of subqueries, several of which share subexpressions. Given the fact that join-intensive workloads like SPARQL queries are expensive to process on MapReduce platforms and query rewritings for inferencing queries are often very large, it is very useful to consider techniques for *MultiQuery Optimization - MQO* for sharing query execution. The challenges of processing SPARQL queries on MapReduce platform arise from the absence of commonly assumed optimization machinery such as database statistics, indexing, etc. Further, traditional cost-based optimization techniques are not MapReduce-aware. More recently, heuristic [3] and rule-based algebraic optimization [4], [5] techniques have been proposed for interpreting and evaluating SPARQL queries more efficiently on MapReduce. A key optimization goal in these contexts is reducing the length of workflow i.e. number of MapReduce cycles due to the materialization, sorting, and network data transfer overhead associated with each cycle.

Techniques for shared execution of SPARQL queries in MapReduce must consider aspects of the MapReduce processing model including, how execution is delineated into a sequence of cycles, the absence of pipelined parallelism and a very limited communication model across execution nodes and so on. For example, a common MQO optimization technique is to factor out common subexpressions and share their results using DAG execution plans. However, this approach is most beneficial with centralized pipelined execution strategies and in fact may lead to longer execution workflows in the MapReduce model. Recently, an algebraic optimization technique [7] that rewrites several SPARQL queries into a single SPARQL query with multiple SPARQL `OPTIONAL` clauses was proposed. This approach also factors out the largest common subpattern into a basic graph pattern while the other clauses are grouped into `OPTIONAL` clauses. For example, the two example graph patterns Q1 and Q2 in Fig. 1(a-b) share a common subpattern corresponding to properties :p1 and :p2, and can be rewritten into a single query with `OPTIONAL` clause as represented as query Q′ in Fig. 1d. The first `OPTIONAL` clause represents triple patterns in Q1 that are not required by Q2, and the second clause represents triple patterns that are only required by Q2 (not in Q1). Another technique for sharing on MapReduce has been proposed in MRShare [6] but only focuses on simple tasks (single cycle workflows) and it is not clear how to extend this approach for execution workflows resulting from SPARQL queries.



```
SELECT * WHERE{
?x :p1 ?a .
?x :p2 ?b .
      OPTIONAL{
        ?x :p3 ?c .
        ?x :p4 ?d .  }
      OPTIONAL{
        ?x :p5 ?e .
        ?x :p6 ?f .  }}
```

Fig. 1: (a-b) example graph patterns Q1 and Q2, (c) optimized graph pattern Q′, (d) SPARQL query corr. to Q′, (e) MR workflow using relational approach

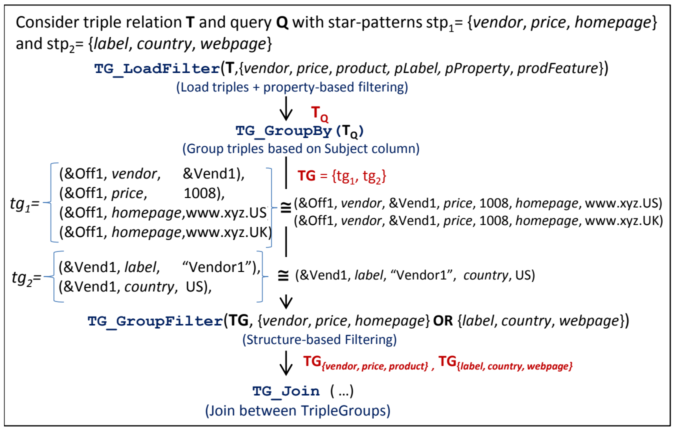

Consider triple relation T and query Q with star-patterns stp1= {vendor, price, homepage} and stp2= {label, country, webpage}
TG_LoadFilter(T,{vendor, price, product, pLabel, pProperty, prodFeature})
(Load triples + property-based filtering)
TQ
TG_GroupBy (TQ)
(Group triples based on Subject column)
TG = {tg1, tg2}
tg1=
(&Off1, vendor, &Vend1),
(&Off1, price, 1008),
(&Off1, homepage,www.xyz.US
(&Off1, homepage,www.xyz.UK)
≅ (&Off1, vendor, &Vend1, price, 1008, homepage, www.xyz.US)
(&Off1, vendor, &Vend1, price, 1008, homepage, www.xyz.UK)
tg2=
(&Vend1, label, "Vendor1"),
(&Vend1, country, US),
≅ (&Vend1, label, "Vendor1", country, US)
TG_GroupFilter(TG, {vendor, price, homepage} OR {label, country, webpage})
(Structure-based Filtering)
TG{vendor, price, product} , TG{label, country, webpage}
TG_Join (…)
(Join between TripleGroups)

Fig. 2: NTGA-based processing of a graph pattern query Q with two star-patterns stp_1 and stp_2 over a triple relation T

We believe that the query rewriting approach presents some unexplored MQO optimization opportunities by enabling it to be addressed at the logical layer using new logical operators. A possibility is devising efficient techniques for Multi-OPTIONAL clauses (analogous in spirit to algorithms for multiway joins). This paper presents a vision for achieving this via a translation to another data model and algebra, *Nested TripleGroup Data Model and Algebra* [4], [5] for efficient evaluation of SPARQL queries on MapReduce.

II. Vision

Fundamentally, NTGA captures starjoins as a "grouping" operation in which groups of triples with the same subject are considered a match for a starjoin pattern if the "*triple-group*" contains triple matches for each branch of a starjoin pattern. Using this approach, multiple starjoins can be executed *concurrently* simply by following a grouping operation over entire (or relevant part of) dataset, with a filter step (in the same MapReduce cycle). The filtering step filters triplegroups based on whether they are matches of *at least one* of the starjoin patterns present in the query. Fig.2 shows part of NTGA's query plan (star-join computation) for query Q with two star subpatterns stp_1 and stp_2. A key operator is the `TG_GroupFilter` which enforces the structural constraints in a star subpattern e.g. tg_1 in Fig.2 is considered a valid match for the set of properties $\{vendor, price, homepage\}$. In contrast, tg_2 violates the constraint $\{label, country, webpage\}$ and is pruned out. Triplegroups produced using this approach are '*content-equivalent*' [4], [8] (denoted as $\cong$) to the set of n-tuples computed using relational joins. e.g.,

$$tg_2 \cong (\text{\&Vend1, label, "Vendor1", country, US})$$

The cost implications for NTGA-based MapReduce processing are shortened execution workflows (relational processing requires 1 cycle per starjoin) and reduced size of intermediate result footprints. The latter is due to the fact that results containing redundant information are represented implicitly in nested data models like NTGA, avoiding the unnecessary overhead of writing, reading, and network data transfer of redundant data.

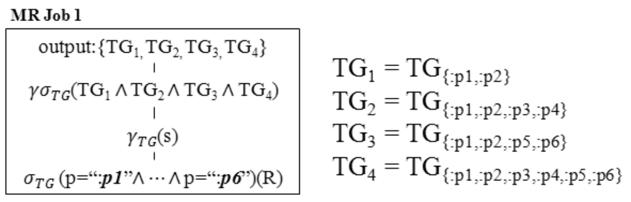

MR Job 1
output:{TG1, TG2, TG3, TG4}
γσTG(TG1 ∧ TG2 ∧ TG3 ∧ TG4)
γTG(s)
σTG (p=":p1"∧ ··· ∧ p=":p6")(R)
TG1 = TG{:p1,:p2}
TG2 = TG{:p1,:p2,:p3,:p4}
TG3 = TG{:p1,:p2,:p5,:p6}
TG4 = TG{:p1,:p2,:p3,:p4,:p5,:p6}

Fig. 3: NTGA-based MR workflow

Our vision is to extend this "multi starjoin" processing to "multi-`OPTIONAL`" processing using extended group filter operator `TG_GroupFilter` ($\gamma\sigma_{TG}$) to support efficient evaluation of queries produced by the MQO `OPTIONAL` query rewriting technique. To illustrate the potential advantages of such an approach, compare the fact that a relational execution plan for query Q$'$ requires 3 MapReduce jobs (MR Job 1, 2, 3) to construct sub stars with properties {:p1,:p2}, {:p3,:p4}, and {:p5,:p6}, respectively. Next, a left outer join (LOJ) links the required and optional subpatterns in an MR job (MR job 4 processes the first `OPTIONAL` pattern, joining $temp1$ and $temp2$ on ?x). In total, 5 MR jobs are required to process Q$'$.

Fig.3 shows the resulting NTGA-based MR workflow. `TG_Filter` (σ_{TG}) selects triples whose properties are listed in the query. `TG_GroupBy` (γ_{TG}) groups triples based on their subjects, generating triplegroups containing different subsets of query-relevant properties. The `TG_GroupFilter` operator selects triplegroups satisfying the structural constraints denoted by `OPTIONAL` patterns, e.g., considering all possible answers for the query Q$'$, the valid answer should match one of the four property groups: (i) {:p1,:p2} (ii) {:p1, :p2, :p3, :p4} (iii) {:p1, :p2, :p5, :p6} and (iv) {:p1, :p2, :p3, :p4, :p5, :p6}. Therefore, `TG_GroupFilter` selects only triplegroups containing such properties, which are TG_1, TG_2, TG_3, and TG_4 in Fig.3. The advantage here is that this filtering of groups does not use any additional MR cycle and is done in the reduce phase of `TG_GroupBy`. Thus, query Q$'$ requires only 1 MR cycle using an NTGA-based query plan as opposed to 5 MR cycles using the relational-style plan. We expect that such savings will lead to more efficient query evaluation.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *Proc. OSDI*, 2004, pp. 10–10.

[2] H. Stuckenschmidt, J. Broekstra, and A. Amerfoort, "Time - space trade-offs in scaling up rdf schema reasoning," in *WISE 2005 Workshops. Volume 3807 of LNCS*. Springer, 2005, pp. 172–181.

[3] M. F. Husain, J. McGlothlin, M. M. Masud *et al.*, "Heuristics-Based Query Processing for Large RDF Graphs Using Cloud Computing," *TKDE*, vol. 23, pp. 1312–1327, 2011.

[4] P. Ravindra, H. Kim, and K. Anyanwu, "An Intermediate Algebra for Optimizing RDF Graph Pattern Matching on MapReduce," in *Proc. ESWC*, 2011, vol. 6644, pp. 46–61.

[5] H. Kim, P. Ravindra, and K. Anyanwu, "From SPARQL to MapReduce: The Journey Using a Nested TripleGroup Algebra," *Proc. VLDB*, vol. 4, no. 12, 2011.

[6] T. Nykiel, M. Potamias, C. Mishra *et al.*, "MRShare: Sharing across Multiple Queries in MapReduce," *Proc. VLDB*, vol. 3, pp. 494–505, 2010.

[7] W. Le, A. Kementsietsidis, S. Duan *et al.*, "Scalable multi-query optimization for sparql," in *ICDE*, 2012, pp. 666–677.

[8] H. Kim, P. Ravindra, and K. Anyanwu, "Scan-sharing for optimizing rdf graph pattern matching on mapreduce," in *IEEE 5th International Conference on Cloud Computing (CLOUD)*, 2012, pp. 139 –146.