Chapter 35
Nesting Strategies for Enabling Nimble MapReduce Dataflows for Large RDF Data

Padmashree Ravindra
North Carolina State University, USA

Kemafor Anyanwu
North Carolina State University, USA

ABSTRACT

Graph and semi-structured data are usually modeled in relational processing frameworks as "thin" relations (node, edge, node) and processing such data involves a lot of join operations. Intermediate results of joins with multi-valued attributes or relationships, contain redundant subtuples due to repetition of single-valued attributes. The amount of redundant content is high for real-world multi-valued relationships in social network (millions of Twitter followers of popular celebrities) or biological (multiple references to related proteins) datasets. In MapReduce-based platforms such as Apache Hive and Pig, redundancy in intermediate results contributes avoidable costs to the overall I/O, sorting, and network transfer overhead of join-intensive workloads due to longer workflows. Consequently, providing techniques for dealing with such redundancy will enable more nimble execution of such workflows. This paper argues for the use of a nested data model for representing intermediate data concisely using nesting-aware dataflow operators that allow for lazy and partial unnesting strategies. This approach reduces the overall I/O and network footprint of a workflow by concisely representing intermediate results during most of a workflow's execution, until complete unnesting is absolutely necessary. The proposed strategies are integrated into Apache Pig and experimental evaluation over real-world and synthetic benchmark datasets confirms their superiority over relational-style MapReduce systems such as Apache Pig and Hive.

DOI: 10.4018/978-1-5225-5191-1.ch035

INTRODUCTION

Data intensive computing applications are increasingly relying on MapReduce (Dean & Ghemawat, 2008) platforms such as Hadoop (Bialecki, Cafarella, Cutting, & O'Malley, 2005), Dryad (Isard, Budiu, Yu, Birrell, & Fetterly, 2007), Hive (Thusoo et al., 2009), Pig (Olston, Reed, Srivastava, Kumar, & Tomkins, 2008) for achieving scalability. Systems such as Hive and Pig provide structured data processing primitives and limited automatic optimization techniques reminiscent of relational database systems. In these systems, a script describes a set of desired data operations (and their relationships) in a high-level language, which is subsequently compiled into a MapReduce workflow whose execution is coordinated over a Hadoop cluster.

Each MapReduce cycle implements the functionality of a subset of operations given in the script. For each such cycle, input is read in from the *Hadoop distributed file system* (HDFS), and partitioned across a set of slave nodes that act as *mappers*. Once the mappers finish execution of their function, a shuffle phase sorts, partitions and stores intermediate map output to local disks. The reducer (slave) nodes contact all mappers, and read and transfer their assigned partitions from the mapper nodes. When the reducer tasks complete, the output is saved back onto the HDFS, from which the next Map phase may read its input. Hence, it is clear that besides the amount of original input data, the amount of data materialized, sorted and transferred over the network during the shuffle phase have a significant impact on the overall costs of processing (Dittrich et al., 2010; Rao, Ramakrishnan, Silberstein, Ovsiannikov, & Reeves, 2012). While some initial data processing operations such as filtering steps reduce the original size of data, some other operations such as the relational join operations are state-producing where outputs could be larger than inputs. Consequently, managing intermediate results in such situations is very important for workloads that are join-intensive.

When processing graph or semi-structured data using relational frameworks, data is typically captured as "thin" relations as $(\mathrm{node}, \mathrm{edge}, \mathrm{node})$ for graph structured data or $(\mathrm{Subject}, \mathrm{Property}, \mathrm{Object})$ in the Semantic Web parlance. The lack of uniform structure in such data makes it harder to model them as "fatter" relations representing a collection of common attributes. The fine-grained model results in the need for multiple join operations for assembling related data.

Systems such as Pig and Hive translate such queries into execution plans in which each cycle processes a set of joins that are on the same column. In graph-oriented view, such joins can be viewed as a star structure and are sometimes called star-joins. Join SJ1 in Figure 1 is a star-join between relations - $\mathrm{T}_{\mathrm{label}}$, $\mathrm{T}_{\mathrm{property}}$, and $\mathrm{T}_{\mathrm{feature}}$, to reassemble tuples related to a product's label, property, and features, respectively. Join J1' is the join linking SJ1 with another star subquery SJ2 . Thus, the MapReduce execution plan for this query will have 3 MR cycles. In general, such plans will have one MR cycle for each star-join, and $(\mathrm{n}$ - 1) MR cycles to join the n star subqueries. For graph-oriented data queries it is not unusual to have n significantly larger than 2. This leads to longer data processing workflows. Given the I/O and network transfer costs associated with each cycle, longer data processing workflows are inherently expensive. Further, the costs compound across cycles for data processing workflows involving multiple state-producing (join) operations.

Additionally, many real-world social network and biological datasets contain multi-valued relationships e.g., millions of Twitter followers of popular celebrities, or multiple references to related proteins in Uniprot datasets. An issue with this in join-intensive processing is that many of the combinations of tuples generated by a join operation contain some redundancy. Specifically, the subtuple containing the