A Cached-Based Stream-Relation Join Operator for Semi-Stream Data Processing

M. Asif Naeem, School of Engineering, Computer and Mathematical Sciences, Auckland University of Technology, Auckland, New Zealand
Imran Sarwar Bajwa, School of Computer Science, University of Birmingham, Birmingham, UK
Noreen Jamil, School of Engineering, Computer and Mathematical Sciences, Auckland University of Technology, Auckland, New Zealand

ABSTRACT

Stream-based join algorithms got a prominent role in the field of real-time data warehouses. One particular type of stream-based joins is a semi-stream join where a single stream is joined with a disk-based relation. Normally the size of this disk-based relation is large enough and cannot be fit into memory available for the join operator. Therefore, the relation is loaded into memory in partitions. A well-known join algorithm called MESHJOIN (Mesh Join) has been presented in the literature to process semi-stream data. However, the algorithm has some limitations. In particular, MESHJOIN does not consider the characteristics of stream data and therefore does not perform well for skewed stream data. This article introduces the concept of caching and based on that presents a novel algorithm called Cached-based Stream-Relation Join (CSRJ). The algorithm exploits skewed distributions more appropriately, and the authors present results for Zipfian distributions of the type that appear in many applications. They test their algorithm using synthetic, TPC-H and real datasets. Their experiments show that CSRJ forms significantly better than MESHJOIN. They also drive the cost model for their algorithm and based on that they tune the algorithm.

KEYWORDS

Performance Optimization, Real-Time Data Warehouses, Semi-Stream Join, Skewed Data

1. INTRODUCTION

Real-time data warehousing plays a prominent role in supporting overall business strategy. By extending data warehouses from static data repositories to active data repositories enables business organizations to better inform their users and to take effective timely decisions (Golfairelli & Rizzi, 2009; Thomsen & Pedersen, 2005; Vassiliadis, 2009). In real-time data warehousing the changes occurring at source level are reflected in data warehouses without any delay. Extraction, Transformation, and Loading (ETL) tools are used to access and manipulate transactional data and then load them into the data warehouse. An important phase in the ETL process is a transformation where the source level changes are mapped into the data warehouse format. Common examples of transformations are units conversion, removal of duplicate tuples, information enrichment, filtering of unnecessary data, sorting of tuples, and translation of source data key.

To explain the transformation phase further we consider an example shown in Figure 1 that implements one of above features, called enrichment. In the example we consider the source updates with attributes product_id, qty, and date that are extracted from data sources. At the transformation...
layer in addition to key replacement (from source key product_id to warehouse key s_key) there is some information added, sales price to calculate the total amount, and the vendor information. In the figure these information with attributes name s_key, s_price, and vendor are extracted at run time from master data and are used to enrich the source updates using a join operator.

In traditional data warehousing the source updates are buffered and join is performed off-line. On the other hand, in real-time data warehousing this operation needs to be performed as the updates are received from the data sources. In implementing the online execution of join, it is observed that due to different arrival rate of both inputs, the transactional or stream input is fast and huge in volume while the master or disk input is slow; the algorithm faces some performance issues due to a bottleneck in the stream of updates.

With the availability of large main memory and powerful cloud computing platforms, considerable computing resources can be utilized when executing stream-based joins. However, there are several scenarios where approaches that can function with limited main memory are of interest. First, the master data may simply be too large for the resources allocated for a stream join, so that a scalable algorithm is necessary. Second, low-resource consumption approaches may be necessary when mobile and embedded devices are involved. For example, stream joins such as the one discussed here could be used in sensor networks. As a consequence, semi-stream join algorithms that can function with limited main memory are important building blocks for a resource-aware system setup.

In the literature, a seminal semi-stream join algorithm MESHJOIN (N. Polyzotis, Skiadopoulos, Vassiliadis, Simitsis, & Frantzell, 2007; Neoklis Polyzotis, Skiadopoulos, Vassiliadis, Simitsis, & 

Figure 1. An example of content enrichment
Lossless Reduction of Datacubes using Partitions
[www.igi-global.com/article/lossless-reduction-datacubes-using-partitions/1821?camid=4v1a](www.igi-global.com/article/lossless-reduction-datacubes-using-partitions/1821?camid=4v1a)

When Spatial Analysis Meets OLAP: Multidimensional Model and Operators
[www.igi-global.com/chapter/when-spatial-analysis-meets-olap/61179?camid=4v1a](www.igi-global.com/chapter/when-spatial-analysis-meets-olap/61179?camid=4v1a)