Optimization of Continual Queries

Sharifullah Khan
National University of Sciences and Technology, Pakistan

INTRODUCTION

Recent advances in the technologies have made it possible to access information from Internet-scale distributed data sources (Florescu, Levy & Mendelzon, 1998). However, finding the right information at the right time is difficult. Update monitoring (Seligman et al., 2000) is a technology that gathers relevant information and forwards it to users in a timely way. Continual queries (CQs) (Chen et al., 2000; Khan & Mott, 2002a; Liu, Pu & Tang, 1999, 2000) provide a significant toolkit for update monitoring. They are persistent queries that are issued once and then are run at regular intervals or when data change until a termination condition is satisfied. They are then removed from the system. They relieve users from having to revisit Web sites or other data sources and reissue their queries frequently to obtain new information that match their queries. A CQ is a typical SQL query having additional triggering and termination conditions. CQs are of two types: change-based and time-based. An example of a CQ is “notify me in the next six months whenever the Microsoft stock price drops by more than 5% from today level.”

CQs OPTIMIZATION

CQs are particularly useful for an environment like the Internet, comprises a large amount of frequently changing information (Chen et al., 2000; Khan & Mott, 2002a; Liu, Pu & Tang, 1999). A CQs system needs to be able to support a large number of queries from a large number of users. This poses many difficulties through the Internet’s widespread distribution and massive use by a large population of users (Chen et al., 2000; Khan & Mott, 2002b). For example, the queries will impact on the local operations of the data source and could overload the data source with duplicate computation of common tasks, each occurring in multiple CQs. Since each data source’s communications and data processing capacity must be divided among all the users as the number of users grows, data servers may become swamped. In addition, these queries could also overload networking with data traffic, much of which may be superfluous. One approach to controlling this problem is to group queries so that they share their computation on the assumption that many queries have a similar structure. For example, consider relations $r_1(a,b,c)$ and $r_2(x,y,z)$. The following queries have similar structure:

$$Q_1 = \sigma_{x=10\land y=x\land z=a} \left( r_1 \ominus r_2 \right)$$
$$Q_2 = \sigma_{x=15\land y=x\land z=a} \left( r_1 \ominus r_2 \right)$$
$$Q_3 = \sigma_{x=5\land y=x\land z=a} \left( r_1 \ominus r_2 \right)$$

Grouping queries optimizes the evaluation of the query by executing common operations in the group of queries just once (Chen et al., 2000; Khan & Mott, 2002b; Roy et al., 2000). Moreover, it also avoids unnecessary query invocation over autonomous data sources on the Internet and reduces data traffic over the networks.

Additionally, after the initial evaluation we want a CQ to return only changes in the data source it queries. The simple and straightforward way to perform this is the complete evaluation approach. This approach uploads the complete results of the CQ rather than just updated results, which increases data transmission over the networks. An important approach to the optimization of CQs is therefore to employ differential evaluation. It means that a CQ is evaluated on the changes that have been made in the base data since its previous evaluation of the CQ after the initial evaluation. This approach reduces the data transmission in the subsequent evaluations of the query (Liu et al., 1996). Clearly, this technique is best suited to conditions where the number of changes is small relative to a much larger quantity of source data (Liu et al., 1996).

Existing Systems

Query grouping on the basis of common operations is not new (Roy et al., 2000). However, the grouping of CQs raises new issues: (1) a CQs system has to handle a large collection of CQs due to the scale of the Internet; (2) CQs are not all made available to the system at the same time; (3) a user’s requests are unpredictable and may change rapidly; and (4) CQs in a group can have different triggering times. So, CQs groups are dynamic; they are continually changing as old queries are deleted and new ones are added. The frequent insertion and deletion of CQs in groups can make those groups inefficient over time and hence reduce overall system performance (Chen et al., 2000; Khan & Mott, 2002b). In this case, one or more