Applied Soft Computing 30 (2015) 72–82

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Robust heuristic algorithms for exploiting the common tasks of relational cloud database queries

Tansel Dokeroglu[a], Murat Ali Bayir[b], Ahmet Cosar[c,*]

[a] Simsoft Computer Technologies, Middle East Technical University, Teknokent Bolgesi, 06800 Ankara, Turkey
[b] Microsoft, 1 Microsoft Way, Redmond, WA 98052, United States
[c] Computer Engineering, Middle East Technical University, 06800 Ankara, Turkey

ARTICLE INFO

Article history:
Received 20 April 2013
Received in revised form 10 August 2014
Accepted 14 January 2015
Available online 7 February 2015

Keywords:
Relational cloud database
Multiple-query optimization
Evolutionary computing
Branch-and-Bound
Hill Climbing

ABSTRACT

Cloud computing enables a conventional relational database system's hardware to be adjusted dynamically according to query workload, performance and deadline constraints. One can rent a large amount of resources for a short duration in order to run complex queries efficiently on large-scale data with virtual machine clusters. Complex queries usually contain common subexpressions, either in a single query or among multiple queries that are submitted as a batch. The common subexpressions scan the same relations, compute the same tasks (join, sort, etc.), and/or ship the same data among virtual computers. The total time spent for the queries can be reduced by executing these common tasks only once. In this study, we build and use efficient sets of query execution plans to reduce the total execution time. This is an NP-Hard problem therefore, a set of robust heuristic algorithms, Branch-and-Bound, Genetic, Hill Climbing, and Hybrid Genetic-Hill Climbing, are proposed to find (near-) optimal query execution plans and maximize the benefits. The optimization time of each algorithm for identifying the query execution plans and the quality of these plans are analyzed by extensive experiments.

1. Introduction

Cloud computing is creating a new market, DataBase as a Service (DBaaS), that has a great potential to attract users ranging from small businesses to very large enterprises seeking high performance solutions. In addition to its high performance, lower cost of ownership, quality of service guarantees, data privacy, scalability, and elasticity are the other opportunities offered by this emerging paradigm.

One of the biggest challenges that DBaaS providers have to cope with is the request of users for continuously meeting the service level agreements. Providing an illusion of infinite resources with increasing database workloads is an NP-Hard optimization problem where the tasks need to be scheduled optimally in order to answer the required services [1,2]. Cloud database query engines can take advantage of common tasks and efficiently manage the resources by using a well-known database optimization technique, Multiple Query Optimization (MQO) [3–8]. Although MQO requires significant search for the identification of common tasks among queries, it has been successfully applied to complex Online Analytical Processing (OLAP) queries that involve big data processing and common tasks [9,10].

MQO has been studied on centralized databases for more than 30 years; however, solving the same problem for relational Cloud databases has not been studied from the perspective of alternative query plans (QP) [8,12,13]. Conventional query engines find the fastest execution plans for single queries and try to execute them as fast as possible on the other hand MQO can execute sets of queries together in shorter times by using their alternative QPs.

In this study, we introduce four robust heuristic algorithms (Branch-and-Bound, Genetic, Hill Climbing, and Hybrid Genetic-Hill Climbing) that improve the total execution time of a set of queries in a relational Cloud database by using alternative QPs that have more sharable tasks. Locality of previously computed tasks and concurrently executing sub-queries are optimized with the proposed robust heuristic algorithms and used for the solution of this problem.

Our contributions in this study can be listed as:

(1) MQO problem is formally adapted for relational Cloud databases including an improved cost model with network communication costs.
(2) Alternative QP generation methods for relational Cloud databases, where the site locations of join tasks can be decided

* Corresponding author. Tel.: +90 505 2370615; fax: +90 312 2105544.
E-mail address: cosar@metu.edu.tr (A. Cosar).

http://dx.doi.org/10.1016/j.asoc.2015.01.026
1568-4946/

intelligently to reduce communication costs, are developed and experimentally evaluated.
(3) Heuristic Branch-and-Bound, Genetic, Hill Climbing, and Hybrid Genetic-Hill Climbing algorithms are developed and experimentally evaluated for solving the Cloud MQO problem.

In Section 2, information about the related work on Grid/Cloud MQO techniques is given. Section 3 gives the formal definition of the problem. Section 4 explains the distributed query engine. Section 5 presents our proposed algorithms that work with alternative QPs. Section 6 discusses the experiments conducted for evaluating the proposed algorithms. Finally our concluding remarks are given in Section 7.

2. Related work

The MQO problem was first defined in 1980s and finding a global optimal QP by using MQO was shown to be an NP-Hard problem [8,16]. A detailed theoretical study of query scheduling, caching, and pipelining in MQO can be found in [18]. Considerable amount of MQO work has been done on relational databases [17,19,20]. The idea of using joint subexpressions has been applied to batch execution of multiple related queries and efficient maintenance of materialized views [49,50]. The studies in [16,51] considered these optimizations and used only the best plans of queries, thus achieving less sharing (i.e. higher total cost) than that could be obtained by using suboptimal QPs. Polat et al. provide heuristics and methods for generating alternative QPs that will improve the performance of MQO [14]. The execution time of a batch of queries is improved by evaluating a common plan task once obtained by using a lightweight and effective mechanism for detecting potential sharing opportunities among QP tasks [46].

When we survey MQO on distributed/parallel databases, we can note early research such as:

- Increasing inter-query locality by decomposing a query into parallel sub-tasks so that a scheduler rearranges the QP tasks execution order for maximizing the reuse of cached-data [21],
- Resource usage models to perform multiple query scheduling on parallel query processing systems in order to reduce the response times of queries [22],
- Dividing a query into sub-queries that can be executed in parallel on many processors and enabling already computed (and cached) sub-query results to be re-used for improving processing speeds of new queries [20].

Mehta and DeWitt developed algorithms to take advantage of intra-operator parallelism, used CPU loads and tuple production rates of *select* and *hash-join* database operations for deciding on the number of allocated processors and the assignment of database operations to these processors [23]. Distributed query processing middleware systems have also been extensively studied as a solution for data intensive scientific applications. MOCHA [25] was one of the first database middlewares developed to execute database queries over distributed data sources. MOCHA could move the code required to process the query to the data storage site. In Beynon [26], user-defined functions can be executed at data storage sites to perform subsetting operations and many *filter* (e.g. aggregation) operators can be run in parallel on a large number of computers.

Indexing the data at each server is an efficient method for distributed query optimization. R-trees are widely used to index and integrate the back-end servers as a single query server. Parallel R-trees, Master R-trees, and Master-Client R-trees are mechanisms used for improving the performance of shared-nothing environments [29]. More specifically, the savings resulting from reusing cached results have to be weighed against the service time and extra storage cost and extra data access load imposed on the server where the cached result is located. Mondal et al. used data migration to shift the workload from heavily loaded servers to lightly loaded servers in shared-nothing environments [30]. Chen et al. considered the network layer of the problem and reduced the communication costs with a query reconstruction algorithm to enable sharing of overlapped data through micro-machines that collaborate for evaluating query batches [6].

IGNITE [11], OGSA-DQP [32], CoDIMS-G [33], and GridDB-Lite are some of the important projects that focus on Cloud/Grid data integration [34]. Except IGNITE, none of these systems has MQO support.

Recently, studies have been performed for adapting traditional query optimizers to Cloud computing. In [47], a classical query optimizer is adapted to Cloud computing workloads where it uses a partitioned database on a shared-nothing architecture. In [44], a parallel data warehouse system optimizer is developed for single queries by considering a rich space of execution alternatives with bushy-tree plans instead of simply parallelizing the best serial plan. Query optimization in Cloud environments can have different goals unlike the traditional query optimizers and the search space becomes much larger. In an interesting study, the scheduling of data processing workflows on the Cloud is considered from the perspective of minimizing the completion time given a fixed budget [48].

Although there exist some initial studies to integrate MQO techniques into existing relational Cloud database query engines, to our knowledge, there is no approach like ours to optimize a batch of queries by employing a relational Cloud database query optimizer which can produce and exploit alternative QPs. Recently, there were two remarkable projects. Giannikis et al. developed a new database architecture that is based on batching queries and shared computation across many concurrent queries in a shared disk, shared L3-cache, multi-core and multi-processor machine [9]. Their model does not try to generate any new alternative plans for input queries. A framework is developed for a Cascade-style Cloud query optimizer to enhance the performance by using MQO techniques for massive data analysis scripts that contain common subexpressions [45] but this approach differs from our technique because new alternative plans are not generated and subexpression costs are used for making optimization decisions only. In our study, a data flow execution model (operator-centric) is used instead of an iterator model, thus most of the mentioned systems cannot be compared with ours.

A distributed query system, IGNITE, that is similar to ours, is chosen for comparing with our system and modifications are done on its architecture to it with the capability of alternative QP generation.

3. Problem formulation

In this section, we formulate the Cloud-MQO problem. A multiple query execution scenario in a relational Cloud database is given, the symbols used throughout the study are explained and the formal problem description is given.

3.1. Sample scenario

A sample relational Cloud database environment for the tpc-h benchmark database can be seen in Fig. 1. Concurrently accessed databases, queries, network, and the sites/processors are the main elements of the Cloud computing environment. In this scenario, there are 6 virtual machines connected via a network.

In Fig. 2, two different QPs are shown for Query 3 of tpc-h database benchmark. QP_1 scans the relations Customer at site S_6,