Brief Announcement: On the Quest of Optimal Service Ordering in Decentralized Queries

Efthymia Tsamoura
Aristotle University of Thessaloniki, Greece
etsamour@csd.auth.gr

Anastasios Gounaris
Aristotle University of Thessaloniki, Greece
gounaria@csd.auth.gr

Yannis Manolopoulos
Aristotle University of Thessaloniki, Greece
manolopo@csd.auth.gr

ABSTRACT

This paper deals with pipelined queries over services. The execution plan of such queries defines an order in which the services are called. We present the theoretical underpinnings of a newly proposed algorithm that produces the optimal linear ordering corresponding to a query being executed in a decentralized manner, i.e., when the services communicate directly with each other. The optimality is defined in terms of query response time, which is determined by the bottleneck service in the plan. The properties discussed in this work allow a branch-and-bound approach to be very efficient.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query Processing*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems

General Terms

Algorithms

Keywords

Web Services, pipelined execution

1. INTRODUCTION

Nowadays, there is a growing interest in systems that are capable of processing complex tasks formulated as Web Service (WS) workflows utilizing remote computational resources. This interest has motivated the development of several scientific and business workflow management systems and WS-oriented systems equipped with DBMS-like capabilities. In the latter systems, data sources and analysis tools expose themselves as WSs, an SQL-like interface for queries involving WSs is provided and the query plans resemble service workflows (e.g., [1]).

Suppose that there are multiple data tuples to be processed by several distinct WSs, each of which resides on a different host and is characterized by different processing speed and selectivity values. Selectivity is defined as the average ratio of output and input tuples and is particularly relevant to scenarios, where selectivity differs from 1, as is the case of filtering services. For example, a WS that receives as input a person's identifier, and returns a list of credit card numbers has average selectivity above one, since there are more credit card numbers than persons. Another service that, for the same input, performs some filtering and returns the input as output only if that person has a good payment history, has average selectivity below one.

In many cases, the order in which the services can be called is flexible in the sense that multiple orderings produce equivalent results. For example, when looking for the credit card numbers of potential customers selecting only those who have a good payment history, the two aforementioned services can be called in any order (see also [1] for a more complex example). However, different orderings may result in significantly different response times. We assume a pipelined execution model, where each WS runs on a different node and the results of one WS may immediately be passed on to the next service in a pipelined fashion. According to the pipelined execution model, the query response time is no longer the sum of the service costs, but is determined by the slowest node [1].

If the services communicate with each other either through an intermediary service or the communication cost incurred by exchanging data between two services is identical for every service pair, then there exists a polynomial algorithm, which produces an optimal ordering in terms of query response time and is applicable regardless of any precedence constraints among the services [1]. This work focuses on a generalization of the aforementioned problem and assumes that the services in the query plan communicate directly with each other following a service choreography approach (i.e., the execution is decentralized), and, in addition, the inter-service communication costs differ. No polynomial time algorithm is known for this more general problem to the best of the authors' knowledge. Also, we believe that such an algorithm is unlikely to exist. The problem we deal with is a generalization of the bottleneck TSP problem, which is known to be NP-hard. More specifically, when (i) setting all service selectivities to 1 and service processing costs to 0, and (ii) considering only the inter-service communication costs, then the optimal service linear ordering problem is the same as the bottleneck TSP one.

In [2], we have proposed a branch-and-bound algorithm that is guaranteed to find the linear ordering of services, which minimizes the query response time and, according to the extensive simulation and real experiments' results appears to be particularly efficient. In this paper, after presenting an overview of our approach in Section 2, we discuss the theoretical underpinnings that allow us to prune the exponential search space effectively.

2. PROBLEM STATEMENT AND BRIEF PRESENTATION OF OUR APPROACH

Assume that all services are selective (i.e., they act as filters), there are no precedence constraints and each service consists of a single thread that processes input tuples and sends output tuples to the next service sequentially; our solution can be applied with minor modifications when these restrictions are relaxed (see [2]), however we examine this restricted case in order to keep the discussion as simple as possible. Let c_i be the average time needed by WS_i to process an input tuple (also referred to as the cost of WS_i), σ_i the selectivity of WS_i and $t_{i,j}$ the time needed to transfer a tuple from WS_i to WS_j. Note that, in practice, tuples are transmitted in blocks [1]; in that case, $t_{i,j}$ is the cost to transmit a block divided by the number of tuples it contains. We also assume that c_i, σ_i and $t_{i,j}$ are constants and independent of the input attribute values and of each other.

A plan $\mathcal{S}$ consists of a linear ordering of the WSs, and its response time is given by the bottleneck cost metric, in accordance to [1]:

$$cost(\mathcal{S}) = \max_{i|WS_i \in \mathcal{S}}((\mathbf{\Pi}_{k|WS_k \in P_i(\mathcal{S})}\sigma_k)(c_i + \sigma_i t_{i,i+1})), \quad (1)$$

where $P_i(\mathcal{S})$ is the set of WSs that are invoked before WS_i in the plan $\mathcal{S}$ and $\mathbf{\Pi}_{k|WS_k \in P_i(\mathcal{S})}\sigma_k$ gives the average number of tuples that reach WS_i per input tuple. The problem we deal with in this work is formulated as follows. Given a set W of N WSs $W = \{WS_0, WS_1, \ldots, WS_{N-1}\}$, where WS is allocated on a host machine, the goal is to construct a linear plan $\mathcal{S}$, which minimizes the bottleneck cost metric given by Eq. (1).

The algorithm in [2] builds on the branch-and-bound optimization approach and is capable of pruning the exponentially large search space effectively. The algorithm starts with an empty plan, to which it appends the less expensive pair of WSs. At each step, the algorithm either appends a service to a partial plan, or prunes the partial plan with a view to exploring additional orderings. The service appended to the partial plan is the less expensive WS with respect to the last service of the partial plan that has not been investigated yet. Two measures, guide this process, ϵ and $\bar{\epsilon}$ respectively. The former corresponds to the bottleneck cost of the partial plan, while the latter is the maximum possible cost that may be incurred by WSs not currently included in the partial plan.

If, for a partial plan $\mathcal{C}$, the condition $\epsilon < \bar{\epsilon}$ is met, this means that the bottleneck cost of the plan beginning with $\mathcal{C}$ depends on the ordering of the services not yet included; so a new WS is appended to $\mathcal{C}$. On the other hand, if condition $\epsilon \geq \bar{\epsilon}$ is met, then the order in which the rest WSs may be appended to $\mathcal{C}$ does not affect its bottleneck cost ϵ, since the maximum possible cost $\bar{\epsilon}$ that may be incurred cannot be higher than ϵ. This implies that the bottleneck cost for all orderings with prefix $\mathcal{C}$ has been established and is set to the current value of ϵ. As such, there is no need to examine plans with prefix $\mathcal{C}$ explicitly. In addition, we prune all plans with prefix the part of $\mathcal{C}$ up to the bottleneck service (including it) from the search space. As explained later, this does not compromise the optimality of the algorithm. Note that in each partial plan $\mathcal{C}$, the bottleneck service may be at any place. After the pruning of the search space, we prune $\mathcal{C}$ up to (without including) the bottleneck service and continue appending new WSs until a new bottleneck cost can be established.

3. SOLUTION PROPERTIES AND PROOF OF CORRECTNESS

In this section, we present the theoretical background of the proposed algorithm, explaining the reasons why the pruning of the search space, although it may be deemed as aggressive, does not compromise the solution's optimality.

LEMMA 1. *If ϵ is the bottleneck cost of a partial plan $\mathcal{C}$, any plan with prefix $\mathcal{C}$ cannot have a lower bottleneck cost.*

This non-decreasing property of ϵ with regards to the size of the partial plan derives directly from Eq. (1).

LEMMA 2. *If for a partial plan $\mathcal{C}$, $\epsilon \geq \bar{\epsilon}$, then, any plan with prefix $\mathcal{C}$ has cost ϵ.*

This derives directly from the definition of these two measures and the fact that selectivities are not greater than 1. If the selectivities may be greater than 1, the way $\bar{\epsilon}$ is computed is slightly modified ([2]).

LEMMA 3. *Let $\mathcal{V}$ be a data structure that contains all the pruned plans up to the bottleneck service (including the latter). Then, no plan $\mathcal{C}$ with prefix any of the plans stored in $\mathcal{V}$ can have bottleneck cost $\epsilon < \rho$, where ρ is the minimum bottleneck cost found so far.*

$\mathcal{V}$ includes prefixes of partial plans up to and including their bottleneck service; the cost of at least one of such bottleneck services is ρ, while the costs of other bottlenecks are either equal to or higher than that. However, for any plan used to construct $\mathcal{V}$, the service appended to the plan just after the bottleneck service during the expansion phase must have been the less expensive one with respect to the service that turned to be the bottleneck service (see the discussion in the previous section), resulting in bottleneck cost $\epsilon \geq \rho$. Because of this policy of choosing the next service to append, any plan $\mathcal{C}$ produced by appending a service to $\mathcal{V}$ has cost $\epsilon \geq \rho$, too. So, with the help of the first lemma, this lemma holds as well.

The algorithm in [2] builds on top of these properties in order to construct the optimal plan quickly. Based on the first lemma, the algorithm can safely exit when there is no WS pair that may form a non-yet investigated prefix of a plan that has a cost lower than the currently lowest bottleneck cost. In general, there are $n!$ orderings, where n is the number of services. However, there are at most $n(n-1)$ prefixes of size two. Also, as the second lemma states, constructing an optimal plan of size n can be reduced to the problem of constructing an optimal plan of a smaller size, while the third lemma drastically reduces the size of the search space, too.

4. REFERENCES

[1] U. Srivastava, K. Munagala, J. Widom, and R. Motwani, "Query optimization over Web Services," in *Proc. of VLDB*, 2006, pp. 355 – 366.

[2] E. Tsamoura, A. Gounaris, and Y. Manolopoulos, "Optimal service ordering in decentralized queries over Web Services," *Technical Report, available from http://delab.csd.auth.gr/ tsamoura/publications.html.*