Optimal Service Ordering in Decentralized Queries Over Web Services

Efthymia Tsamoura, Aristotle University of Thessaloniki, Greece

Anastasios Gounaris, Aristotle University of Thessaloniki, Greece

Yannis Manolopoulos, Aristotle University of Thessaloniki, Greece

ABSTRACT

The problem of ordering expensive predicates (or filter ordering) has recently received renewed attention due to emerging computing paradigms such as processing engines for queries over remote Web Services, and cloud and grid computing. The optimization of pipelined plans over services differs from traditional optimization significantly, since execution takes place in parallel and thus the query response time is determined by the slowest node in the plan, which is called the bottleneck node. Although polynomial algorithms have been proposed for several variants of optimization problems in this setting, the fact that communication links are typically heterogeneous in wide-area environments has been largely overlooked. The authors propose an attempt to optimize linear orderings of services when the services communicate directly with each other and the communication links are heterogeneous. The authors propose a novel optimal algorithm to solve this problem efficiently. The evaluation of the proposal shows that it can result in significant reductions of the response time.

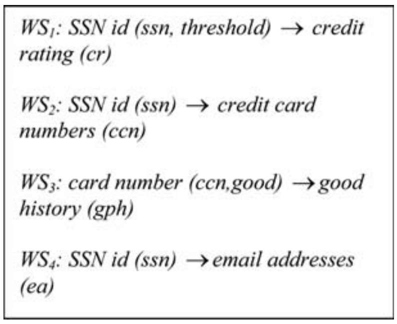
Keywords: Distributed Data Management, Heterogeneous, Optimization, Predicates, Web Services

INTRODUCTION

Technologies such as grid and cloud computing infrastructures and service-oriented architectures have become adequately mature and have been adopted by a large number of enterprises and organizations. This trend has altered, to an extent, the way complex computational tasks are formulated, giving rise to approaches that rely on composition of services to be executed in a parallel and distributed manner (Alpdemir et al., 2004; Ng, Ooi, Tan, & Zhou, 2003; Malik, Szalay, Budavari, & Thakar, 2003). As a consequence, there is a growing interest in systems that are capable of processing complex tasks formulated as Web Service (WS) workflows utilizing remote computational resources. These tasks may involve the complete management of online sales, the cleaning of large volumes of accumulated data from mistypes, incorrect entries, etc., and the loosely-coupled integration of local applications with tools made available on the Web.

DOI: 10.4018/ijkbo.2011040101

Figure 1. Services of example 1



WS_1: SSN id (ssn, threshold) $\rightarrow$ credit rating (cr)

WS_2: SSN id (ssn) $\rightarrow$ credit card numbers (ccn)

WS_3: card number (ccn,good) $\rightarrow$ good history (gph)

WS_4: SSN id (ssn) $\rightarrow$ email addresses (ea)

In Srivastava, Munagala, Widom, and Motwani (2006), the notion of Web Service Management System (WSMS) is introduced as a general purpose system, which possesses such advanced processing capabilities. In a WSMS, processing of data takes place through (remote) calls to WSs. The latter provide an interface of the form $WS : X \rightarrow Y$, where *X* and *Y* are sets of attributes, i.e., given values for attributes in *X*, *WS* returns values for the attributes in *Y*, as shown in the following example adapted from Srivastava et al. (2006). In the generic case, the input data items (or tuples) may have more attributes than *X*, while attributes in *Y* are appended to the existing ones. Note that in the rest of this article, we will use the terms tuple and data item interchangeably.

Example 1. Suppose that a company wants to obtain a list of email addresses of potential customers selecting only those who have a good payment history for at least one card and a credit rating above some threshold. The company has the right to use the WSs listed below (Figure 1) that may belong to third parties. The input data containing customer identifiers is supplied by the user.

There are multiple valid orderings to perform this task, although there is one precedence constraint: WS_2 must precede WS_3. The optimization process aims at deciding on the optimal (or near optimal) ordering under given optimization goals. Two possible WS linear orderings that can be formed using the above services are $C_1 = WS_2\ WS_3\ WS_1\ WS_4$ and $C_2 = WS_1\ WS_2\ WS_3\ WS_4$. In the first ordering, first, the customers having a good payment history are initially selected (WS_2, WS_3), and then, the remaining customers whose credit history is below some threshold are filtered out (through WS_1). The C_2 linear plan performs the same tasks in a reverse order. The above linear orderings have different response time. In a subsequent section it will be shown that C_2 is the optimal one.

Optimizing the order of WS calls in a workflow is an important problem that arises in many business and e-science problems. Another example taken from bioinformatics is presented in Craddock, Lord, Harwood, and Wipat (2006), where, given a set of proteins taken from twelve Bacillus bacterium species, the goal is firstly to classify the secreted proteins and then to analyze them through clustering. The analysis of Bacillus bacteria is crucial not only due to their industrial usages in the production of enzymes and pharmaceuticals, but also because of the diversity of their exposed characteristics; the Bacillus genus includes species that are capable of promoting plant growth and producing antibiotics, as well as harmful bacteria such as the Bacillus anthracis. The goal of analysis is to identify families of Bacillus bacteria with respect to the proteins that they synthesize. In such a workflow, when the order of the performed checks for identifying the kind of a protein (e.g., lipoprotein) is modified, the workflow efficiency changes, too.

In many cases, the problem of optimizing queries over WSs becomes equivalent to that of an optimal ordering of a query's WS calls. Usually, the goal of the optimization is to minimize the response time of queries, even though there may be other metrics of interest, such as total time, monetary cost and aggregate resource utilization. In this work, we focus on the minimization of query response time, exclusively. Other problems of high significance for industrial applications, such as the minimization of detection time of malfunctioning components based on pipelined tests can be easily reduced to the same problem.

In one aspect, the problem of optimal ordering of WS with a view to minimizing the response time may resemble the problem of ordering commutative filters in pipelined queries with conjunctive predicates (Hellerstein & Stonebraker, 1993; Krishnamurthy, Boral, & Zaniolo, 1986), in the sense that the calls to WSs may be treated in the same way as expensive predicates. Ordering some types of relational joins can be reduced to the same problem, as well (Babu, Motwani, & Munagala, 2004). As such, service-based queries can benefit from database technology, and database solutions have partially inspired our approach to optimizing service-based queries, as well. However, there are also many substantial differences between service-based and traditional database queries given that there may exist precedence constraints between the WSs, selectivities may be higher than 1 (e.g., WS_2 in the example) and, typically, the execution of queries over WSs takes place in a both distributed and parallel manner. More specifically, each WS is executed on a different node and the results of one WS may immediately be passed on to the next service in a pipelined fashion, so that the tuples already processed by a WS are processed by the subsequent WS in the plan at the same time as the former processes new input tuples.

According to the pipelined execution model, when no data items (or tuples) are dropped or new tuples are generated, the maximum rate at which input tuples can be processed through a single plan equals the minimum processing rate of all services; the corresponding service, i.e., the WS that spends, on average, the most time per input tuple, is termed the bottleneck WS. This model imposes new optimization challenges. For example, the query response time is no longer the sum of the cost in time units of all the WSs in the pipelined plan, but is determined by the slowest node (Condon, Despande, Hellerstein, & Wu, 2009; Despande & Hellerstein, 2008; Srivastava et al., 2006).

The problem of minimizing the bottleneck cost has received significant attention recently. In Srivastava et al. (2006), along with the introduction of WSMSs, an efficient optimization algorithm is presented that considers precedence constraints among the WSs. Another characteristic of this work is that it can deal with any selectivity values and build plans where the output of a service is fed to multiple services simultaneously. The proposals in Condon et al. (2009) and Despande and Hellerstein (2008) introduce faster algorithms that maximize the data flow by defining the set of interleaving plans along with the proportion of tuples routed to each plan in order to maximize the aggregate processing rate; nevertheless, all the plans are linear, i.e., each WS has at most one input service and one output. More detailed discussion of the related work is deferred in a subsequent section. However, a common feature of all these algorithms is that they do not take the potentially heterogeneous communication links between the services into account, which is significant when the execution is decentralized given also that the communication cost may be the dominant cost. This is in line with the WSMS in Srivastava et al. (2006), which assumes that the output of a service is fed to the subsequent service indirectly through a central management component thus annihilating the need to consider the different communication costs explicitly. As such, in existing proposals, the bottleneck cost depends solely on the service costs and selectivities, instead of taking also into account the inter-service communication cost, especially when the execution occurs in a wide are environment. In the previous example, the optimal ordering may differ when all services are at a single place, when they are all at a different place, and when, for instance, only WS_1 and WS_5 are co-located.

The main contribution of this work is that it addresses the afore-mentioned limitation by proposing an efficient algorithm for the single optimal ordering of services when the services communicate directly with each other and the communication costs between the services may differ. Our algorithm is based upon the branch and bound optimization paradigm and operates regardless of any precedence constraints. It can be deemed as an extension to Srivastava et al. (2006) accounting for decentralized execution, except that we are only interested in linear orderings. The evaluation of the proposal shows that it is efficient and can result in significant reductions of the response time up to an order of magnitude in many realistic scenarios.

The remainder of this article is structured as follows. The problem we deal with is formally introduced in the following section. After that, we present the algorithm in detail discussing its main concept, correctness proofs, and implementation issues. The evaluation section discusses the performance benefits of our approach. Finally, we discuss the related work on modern query optimization problems, as well as extensions of the proposed algorithm and future work.

PROBLEM FORMULATION

In our parallel execution model, each WS runs on a different node in a separate thread that processes input tuples and sends output tuples to the next service sequentially; our solution however can be applied to the case when separate threads are responsible for data processing and transmission in a straight-forward manner. 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[1]. The selectivity is defined as the average ratio between the number of output and input data items (i.e., tuples); if the selectivity is below one, the service acts as a filter, whereas, if the selectivity is above one, the service leads to increase of the data volume. We assume that c_i, σ_i and $t_{i,j}$ are constants and independent of the input attribute values. We further assume that the selectivities of WSs are independent of each other, and, in the generic case, can be greater than 1. A constrained service WS_i has at least one prerequisite service WS_j, which is denoted as $WS_j \prec WS_i$. The precedence constraints are part of the input (e.g., in the form of a DAG).

A plan S consists of a linear ordering of the WSs, which respects any precedence constraints and its response time is given by the bottleneck cost metric, in accordance to (Srivastava et al., 2006):

$$\cos t(S) = \max_{i|WS_i \in S} \left\{ \prod_{k|WS_k \in P_i(S)} \sigma_\kappa (c_i + \sigma_i t_{i,i+1}) \right\}, \tag{1}$$

where $P_i(S)$ is the set of WSs that are invoked before WS_i in the plan S. Throughout the paper we will refer to $T_{i,j} = c_i + t_{i,j}\sigma_i$ as the aggregate cost of WS_i with respect to WS_j. If $t_{i,j}$ is equal for all service pairs, the problem can be solved in polynomial time, as shown in Srivastava et al. (2006). Here we deal with the generic --and more realistic-- case, where $t_{i,j}$ may differ, for which to the best of our knowledge, there is no polynomial solution. More specifically, the problem we deal with in this work is formulated as follows:

Problem Formulation: Given a set W of N WSs $W = \{WS_0, WS_1, \ldots, WS_{N-1}\}$, where each one of them is allocated on a host machine, find the linear plan S, which minimizes the bottleneck cost metric given by Eq. (1)[2].

OPTIMAL LINEAR PLAN CONSTRUCTION ALGORITHM

The proposed algorithm is based on the branch-and-bound optimization approach. Let $C=WS_{k(0)}\, WS_{k(1)} \dots WS_{k(n)}$ be a partial linear plan. *k* is used throughout the paper to denote the mapping from positions in the WSs to the indices of the WSs at those positions.

Two cost metrics guide the plan building process, ε and ε' respectively. The former corresponds to the bottleneck cost of *C*, while the latter is the maximum possible cost that may be incurred by WSs not currently included in *C*. Cost ε' is utilized to speed up the algorithm. For simplicity, we will first discuss the case where the selectivities are not greater than 1. ε and ε' are given by the following equations:

$$\varepsilon = \max\left\{T_{k(0),k(1)}, \max_{1\le i<n}\left\{\left(\prod_{j=0}^{i-1}\sigma_{k(j)}\right)T_{k(i),k(i+1)}\right\}\right\}, \tag{2}$$

$$\varepsilon' = \max_{l,r}\left\{\begin{matrix}\left(\prod_{j=0}^{n}\sigma_{\kappa(j)}\right)T_{l,r}, WS_l \notin C \\ \left(\prod_{j=0}^{n-1}\sigma_{\kappa(j)}\right)T_{l,r}, WS_l = WS_{k(n)}\end{matrix}\right\}, \tag{3}$$

$WS_r \notin C$ in both cases.

The algorithm proceeds in two phases, namely the *expansion* and the *pruning* one. During expansion, new WSs are appended to a partial plan *C*, while during the latter phase WSs are pruned from *C* with a view to exploring additional orderings. If, for a partial plan *C*, the condition ε< ε' is met, this means that the bottleneck cost of the plan beginning with *C* depends on the ordering of the services not yet included; so a new WS_r is appended to *C*. On the other hand, if condition $\varepsilon \geq \varepsilon'$ is met, then the order in which the rest WSs may be appended to *C* does not affect its bottleneck cost ε, since the maximum possible cost ε' that may be incurred cannot be higher than ε. In that case, the linear plan *C* is essentially a solution and the pruning step is triggered. Partial plans are also pruned when they cannot form a prefix of an optimal solution. Let $C=WS_{k(0)}\, WS_{k(1)} \dots WS_{k(n)}$, where $0 \leq n < N$ and $WS_{k(i)}$ be the bottleneck WS of *C*, where $0 \leq i \leq n$. Then *C* is pruned as follows:

$$C = \begin{cases} \varnothing & , i = 0 \\ WS_{k(0)}WS_{k(1)} \dots WS_{k(i-1)}, & 0 < i \leq n \end{cases} \tag{4}$$

To further improve the efficiency of the algorithm, the prefixes up to the bottleneck service $WS_{k(i)}$ (denoted as *C'*) of the plans for which the expansion phase has been completed are stored in a list *V*. To avoid investigating the same solutions multiple times, the function $\pi(X)$ is employed, where *X* is a (potentially partial) plan. This function returns *true* if no WS plan stored in *V* is prefix of *X*. As will be discussed later, the plans considered must satisfy the $\pi()$ function thus yielding better running times without compromising the optimality of the algorithm.

Detailed Algorithm Description

The complete algorithm is shown in Figure 2, where *S* denotes the best linear plan found so far and ρ its bottleneck cost. In every iteration of the algorithm, the cost measures ε and ε' are evaluated. *F(C)* is the set of all WSs for which all the prerequisite WSs have already been added to *C*. More formally *F(C)* is given by:

$$F(C) = \left\{WS_i \mid WS_i \notin C \wedge M_i \subseteq C\right\} \tag{5}$$

and M_i is the set of all WSs that must appear before WS_i, i.e., $M_i = \left\{WS_j \mid WS_j \prec WS_i\right\}$. Obviously, for unconstrained services, $M_i = \varnothing$.

There are three cases depending on the values of ε, ε′ and ρ in a partial plan *C*:

$\varepsilon < \varepsilon', \varepsilon < \rho$ (lines 12-24): if *C* is empty, e.g., it is the initial iteration, the algorithm searches for the most promising WS pair that

Figure 2. The proposed algorithm

```
Inputs
  N: Number of input WSs
  T: An N × N matrix, where $T_{i,j} = c_i + t_{i,j}\ \sigma_i$
Outputs
  S: optimal linear ordering

  1. $S \leftarrow \varnothing$; {S is the current best linear plan}
  2. $\rho \leftarrow \infty$; {$\rho$ is the bottleneck cost of S}
  3. $V \leftarrow \varnothing$; {V is a list of partial linear plans}
  4. $C \leftarrow \varnothing$;
  5. $F(C) \leftarrow \{WS_i \mid M_i = \varnothing\}$;
  6. while true do
  7. Estimate $\varepsilon$ using Eq.(2);
  8. Estimate $\varepsilon'$ using Eq.(3);
  9. if $\varepsilon \geq \rho$ then
  10. V.push($C'$);
  11. Trim C following Eq.(4);
  12. else if $\varepsilon < \varepsilon' \wedge \varepsilon < \rho$ then
  13. if $C = \varnothing$ then
  14. Find the services $WS_l$ and $WS_r$ using Eq.(6);
  15. $C = WS_l\ WS_r$;
  16. else if $C \neq \varnothing$ then
  17. Find a $WS_r$ using Eq.(7);
  18. If no such WS can be found then
  19. Trim C using Eq.(4), where the bottleneck WS is set to $WS_{k(n)}$;
  20. V.push($C'$);
  21. else
  22. Append the $WS_r$ to C;
  23. end if
  24. end if
  25. else if $\varepsilon' \leq \varepsilon < \rho$ then
  26. V.push($C'$);
  27. $S \leftarrow C$;
  28. Trim C following Eq.(4);
  29. $\rho \leftarrow \varepsilon$;
  30. end if
  31. Update F(C) using Eq. (5);
  32. If Termination Condition is true then
  33. return S;
  34. end if
  35. end while
```

has not been considered yet. Such a WS pair must satisfy the following equation

$$T_{l,r} = \min_{i,j|M_i=\varnothing\wedge M_j\subseteq\{WS_i\}\wedge\pi(WS_iWS_j)=true}\left\{T_{ij}\right\} \quad (6)$$

If the partial plan is not empty, the algorithm searches for a new WS WS_r, such that:

$$T_{k(n),r} = \min_{WS_j\in F(\mathcal{C})\wedge\pi(\mathcal{C}WS_j)=\text{true}}\left\{T_{k(n),j}\right\} \quad (7)$$

These sub-cases comprise the expansion case. In case where no WS_r can be found, the last WS of *C* is pruned (lines 18-20) to support cases in which no service can be appended, e.g., all plans with prefix *C* have been examined. Note that the condition $\varepsilon < \rho$ is considered along with $\varepsilon < \varepsilon'$, because it is worth performing the expansion phase only for plans with $\varepsilon < \rho$.

- $\varepsilon' \leq \varepsilon < \rho$ (lines 25-30): the current best linear plan *S* is set to *C* and its bottleneck cost ρ is updated. Condition $\varepsilon \geq \varepsilon'$ ensures that the bottleneck cost of *C*, which is lower than the current lowest cost, will not increase if new WSs are appended. After that, *C* is pruned following the Eq. (4) and the algorithm continues. The intuition behind Eq. (4) is as follows. $WS_{k(i+1)}$ in Eq. (4) satisfies either Eq. (6), or (7), i.e., it is the WS such that $WS_{k(i)}$ has the minimum cost $T_{k(i),k(i+1)}$. Thus, the cost that may be incurred by any other WS appended to $WS_{k(i)}$, will be higher than the current bottleneck cost, i.e., it is worthless to investigate plans with prefix $WS_{k(0)} \ldots WS_{k(i)}$.
- $\varepsilon \geq \rho$ (lines 9-11): *C* cannot yield an optimal solution, since its bottleneck cost is higher than the bottleneck cost of *S*. Thus, *C* is pruned following Eq. (4).

The algorithm can safely terminate when the less expensive pair of WSs satisfying the π function cannot improve the current bottleneck cost, which means that the best possible linear plan not yet visited has at least as high bottleneck cost ε as ρ.

Termination Condition: Let ρ be the minimum bottleneck cost found so far. The algorithm terminates, when there are two services WS_l and WS_r such that:

$$T_{l,r} \geq \rho, M_l = \varnothing \wedge M_r \subseteq \{WS_l\} \wedge \pi(WS_lWS_r) = \text{true} \quad (8)$$

Thus far, we have discussed the case when service selectivities are not higher than 1. If there exist $\sigma_i > 1$, then the same algorithm is still valid; the only change is in the way ε' is computed in Eq. (3). More specifically, ε' in Eq. (3) is multiplied by the product of all $\sigma_i > 1$ such that $WS_i \notin C$. The proof of the algorithm's correctness is in a subsequent section.

An Example of the Algorithm

We continue with Example 1. Table 1 shows the per tuple processing costs and the selectivity values of the services introduced in Example 1, while Table 2 shows the inter-service communication costs. Note that σ_2=1.5 and σ_4=2.5 mean that every customer has, on average, 1.5 credit cards and 2.5 email addresses, respectively. In this example, it is assumed that the cost spent to transfer input data to a service is negligible. However, if this hypothesis does not hold, we can realize the different data communication costs by using a source service WS_0 with zero processing cost and σ_0=1. That service is considered to be the source of input data. In Figure 3, the partial plans at the end of every of the iterations are shown. Before presenting the steps of the algorithm, recall that WS_3 cannot called prior to WS_2.

Initially, $\varepsilon = 0 < \varepsilon' = \sigma_4 T_{2,1} = 87.5$, $\varepsilon < \rho$ ($\rho = \infty$) and $C = \varnothing$. The algorithm starts by identifying the WS pair, which incurs the minimum cost; see lines 13-15 of Figure 2. The corresponding WSs are WS_1 and WS_4. After that, $C=WS_1WS_4$. In the second iteration, since $\varepsilon = 3.6 < \varepsilon' = \sigma_1 \times T_{4,3} = 5.4$,

Table 1. Processing cost and selectivities of the services presented in example 1

WS_i	1	2	3	4
Cost of WS_i (c_i)	2	5	3	4
Select. of WS_i (σ_i)	0.1	1.5	0.3	2.5

Table 2. Communication cost values of the services presented in example 1

$WS_i \setminus WS_j$	1	2	3	4
1	-	20	18	16
2	20	-	9	15
3	18	9	-	20
4	16	15	20	-

$\varepsilon < \rho = \infty$ and $C \neq \varnothing$, a new WS is appended to *C*, which must satisfy the Eq. (7); that service is WS_2. In the third iteration, since $\varepsilon = 4.15 < \varepsilon' = \sigma_1 \times \sigma_4 \times T_{2,3} = 4.625$, $\varepsilon < \rho = \infty$ and $C \neq \varnothing$, the service WS_3 is appended to *C* forming the partial plan $C{=}WS_1 WS_4 WS_2 WS_3$. Now, since $\varepsilon = 4.625 > \varepsilon' = 0$ and $\varepsilon < \rho = \infty$, a solution is found. According to steps in lines 25-30 of Figure 2, *S* is set to *C*, $\rho = 4.625$ and *C* is pruned following the Eq. (4). After the pruning, $C = WS_1WS_4$, since the bottleneck WS is WS_2. The termination condition is not triggered given that there exists a two service prefix that satisfies π and its cost is lower than ρ : $T_{1,2} = 4$.

In the fifth iteration, since $\varepsilon = 3.6 < \varepsilon' = 5.4$, $\varepsilon = 3.6 < \rho = 4.625$ and $C \neq \varnothing$, a new WS needs to be appended to *C*. However, since such a service cannot be found (WS_3 cannot be appended to *C* prior to WS_2), *C* is pruned after setting the bottleneck service to WS_4; see lines 18-20 of Figure 2. After the pruning, $C{=}WS_1$. In the sixth iteration, since $\varepsilon = 0 < \varepsilon' = 8.1$, $\varepsilon = 0 < \rho = 4.625$ and $C \neq \varnothing$, WS_2 is appended to *C*. In the seventh iteration, a new service is also appended to *C* forming the partial plan $C{=}WS_1WS_2WS_3$. In the eight iteration, a new solution is found since $\varepsilon = 4 > \varepsilon' = 1.35$ and $\varepsilon < \rho = 4.625$. Thus, $S = WS_1\, WS_2\, WS_3$, $\rho = 4$ and C is pruned according to Eq. (4). Now, the termination condition is triggered, since the cost of the less expensive WS pair satisfying π, which is WS_2 WS_3 (the services' pair WS_1 and WS_3 cannot be placed at the beginning of a plan, since WS_3 has WS_2 as a prerequisite service), is now higher than ρ : $T_{2,3} = 18.5 > \rho = 4$.

Proof of Correctness

Lemma 1. If ε is the bottleneck cost of C, any plan with prefix C cannot have a lower bottleneck cost.

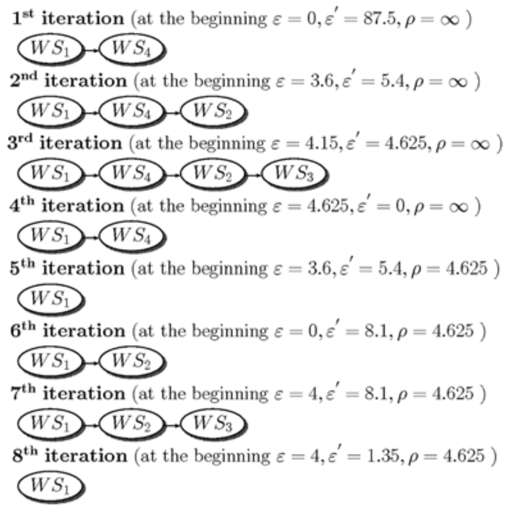
This non-decreasing property of ε with regards to the size of the partial plan derives directly from Eq. (1). Also, as explained in the previous subsections, the following lemma holds:

Lemma 2. If for a partial plan C, $\varepsilon \geq \varepsilon'$, then, any plan with prefix C has cost ε.

This derives from Eq. (2) and (3) and the fact that selectivities are not greater than 1.

Lemma 3. No plan C with prefix any of the plans stored in V can have bottleneck cost

Figure 3. An example of the proposed algorithm using the services presented in example 1


1st iteration (at the beginning $\varepsilon = 0, \varepsilon' = 87.5, \rho = \infty$)
WS_1 WS_4
2nd iteration (at the beginning $\varepsilon = 3.6, \varepsilon' = 5.4, \rho = \infty$)
WS_1 WS_4 WS_2
3rd iteration (at the beginning $\varepsilon = 4.15, \varepsilon' = 4.625, \rho = \infty$)
WS_1 WS_4 WS_2 WS_3
4th iteration (at the beginning $\varepsilon = 4.625, \varepsilon' = 0, \rho = \infty$)
WS_1 WS_4
5th iteration (at the beginning $\varepsilon = 3.6, \varepsilon' = 5.4, \rho = 4.625$)
WS_1
6th iteration (at the beginning $\varepsilon = 0, \varepsilon' = 8.1, \rho = 4.625$)
WS_1 WS_2
7th iteration (at the beginning $\varepsilon = 4, \varepsilon' = 8.1, \rho = 4.625$)
WS_1 WS_2 WS_3
8th iteration (at the beginning $\varepsilon = 4, \varepsilon' = 1.35, \rho = 4.625$)
WS_1

$\varepsilon < \rho$, where ρ is the minimum bottleneck cost found so far.

The plans in *V* fall into two categories. Firstly, *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 higher. However, for any plan used to construct *V*, the service appended to the plan just after the bottleneck service during the expansion phase must have satisfied either Eq. (6), or (7), resulting in bottleneck cost $\varepsilon \geq \rho$. Because of Eq. (6) and (7), any plan *C* produced by appending a service to *V* has cost $\varepsilon \geq \rho$, too. So, with the help of the first lemma, this lemma holds as well. Secondly, *V* includes partial plans for which no new services can be appended, due to either precedence constraints or the fact that all combinations have been already explored (line 20 of Figure 2). The former subcase cannot lead to any solutions, whereas the latter is similar to the first case. This completes the proof of the lemma, and also shows the correctness of using the $\pi()$ function, which builds upon this lemma.

Theorem 1. The algorithm finds the optimal solution.

A sketch of the proof is as follows. In order to prove the correctness of the algorithm, we must prove that all possible orderings have been checked, either directly or indirectly, and the termination condition is correct and the output is a valid solution. The algorithm, with the help of the $\pi()$ function does not explore plans the prefix of which is stored *V*. We have proven that this does not compromise its optimality. The algorithm exits when any WS pair that is a valid beginning of a plan does not have a cost lower than the currently lowest bottleneck cost. In general, there are *n!* orderings, where *n* is the number of services; of course this can be reduced due to precedence constraints. However, there are at most *n(n-1)* prefixes of size two. If all these prefixes satisfy the termination condition, with the help of the first lemma we can show that all *n!* orderings cannot have a bottleneck cost that is lower than ρ. Finally, *S* in Figure 2 is a partial plan, but, as the second lemma shows, any plan with prefix *S* can form a complete optimal ordering.

Note that the termination condition can be reduced to simpler statements in some specific cases. For example, when there are precedence constraints such that only a single WS can be at position *0*, e.g., it is the service responsible for source data generation, then the termination condition can be reduced to the statement that the algorithm can exit if the bottleneck position reaches the first position $WS_{k(0)}$. The proof of correctness is similar to the case when WSs are selective.

Implementation Issues

In the current subsection, we will discuss some implementations issues. We have employed some simple data structures in order to speed up the execution of expensive operations, namely the identification of the next WS to be appended to a partial plan (Eq. (7)), the evaluation of ε' and the check of the termination condition. Note that the computation of the bottleneck cost ε of *C* and the detection of the bottleneck service require linear time; also $\pi()$ can be efficiently implemented with the help of a prefix tree.

To speed up the identification of the next WS, a preprocessing step takes place before the algorithm execution. According to this step, for each service WS_i a doubly-linked list L_i of all services WS_j in increasing order of the corresponding aggregate cost $T_{i,j}$ is constructed. Thus, the next WS to be added after $WS_{k(n)}$ is found using the following simple search approach: we start from the head of list $L_{k(n)}$. If the current WS is not included in *C* and all its prerequisites services are included in *C*, then the desirable WS is found, otherwise the search continues. The computation of ε' is performed using an analogous approach. For every WS_i that it is either the last WS of *C* or does not belong to *C*, the search starts from the end of L_i, since the maximum possible process/transfer cost must be found. The search in every L_i stops when the first WS not currently included in *C* is found and the maximum aggregate cost among those found in every L_i is returned.

Finally, the check of the termination condition can be efficiently done with the help of a min heap. The contents of this heap vary depending on whether the WSs are constrained or not. In the former case, the min heap contains all WS couples (WS_i WS_j), while in the latter case; it contains only the couples that can constitute a valid prefix of a plan. The values of the heap are the costs $T_{i,j}$ of the corresponding couples. Every time we check whether the termination condition is met, the root of the min heap is accessed and the plan $X=WS_i\ WS_j$ is formed from the WS couple (WS_i WS_j) stored in the root of the min heap. If the partial plan *X* satisfies the $\pi(X)$, then the corresponding cost is kept. Otherwise the root of the min heap is deleted and the next new root element is accessed. From the above, it follows that the complexity of this operation is *O(log k)*, where $k \leq N^2$, while the complexity of naive approach is $O(N^2)$.

EVALUATION

In order to study the performance and the efficiency of the proposed algorithm, we have conducted several experiments using simulations of wide-area environments. The performance of the algorithm was evaluated through the comparison of the response times of WS plans built by the proposed algorithm and the Greedy algorithm in (Srivastava et al., 2006), while the efficiency was measured in terms of the absolute time needed to construct the plans. Intuitively, the performance and the efficiency of the proposed algorithm are affected by various parameters. However, we concentrate on three of them, namely the number of input services, the ratio between processing and transferring costs of tuples and the network heterogeneity.

In order to compare the performance of the proposed algorithm and the Greedy one when the number of input services varies, we produce a set A of twenty-five simulations instances, $A = \{a_N \mid N \in \{10, 20, \ldots, 250\}\}$, where each simulation instance a_N consists of

N services. The services' processing costs follow a Gaussian distribution with mean value 10 and standard deviation 2. The communication costs $t_{i,j}$ between the services follow a Gaussian distribution, too, with mean value 25 and standard deviation 2.5. That means that, on average, the cost needed to transfer a tuple will be 2.5 times higher than the cost spent to process it. Finally, the selectivity of the services is uniformly distributed between 0 and 1. The proposed algorithm and the Greedy one are executed on each simulation instance. Let ρ_{α_N} and ρ'_{α_N} be the corresponding response times of the plans built by the aforementioned algorithms for a simulation instance a_N. Figure 4 (left bars) shows the ratio $\rho'_{\alpha_N} / \rho_{\alpha_N}$ for $N \in \{10, 20, \ldots, 250\}$. We observe that the proposed algorithm can yield significant performance improvements of several factors (up to four). Secondly, as the number of input services increases, the performance of the proposed algorithm increases and the response time deviations between the Greedy plans and the optimal ones (built by the proposed algorithm) increase. For example, the plan built by Greedy for the a_{200} setting has 4 times higher response time than the plan built by the proposed algorithm. This value decreases to 1.25 for the a_{10} simulation instance.

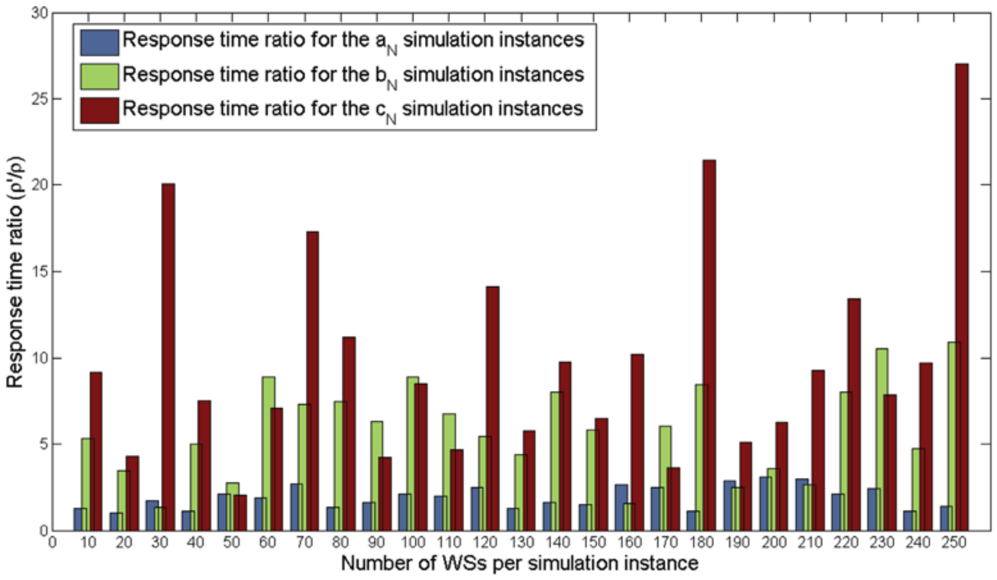
Now, we turn our attention to two other parameters, namely the ratio between the processing and the transferring cost of tuples and the network heterogeneity. To this end, we produce two new sets of twenty-five simulation instances B and C following a procedure similar to the one described above. We vary only the mean transferring cost per tuple and the network heterogeneity. The network heterogeneity is controlled through the standard deviation of the communication cost values distribution; as the standard deviation increases, the network heterogeneity becomes higher. In the simulation instances' set B, the communication cost values follow a Gaussian distribution with mean value 200 and standard deviation 40, while in the third set of simulation instances, the mean transferring cost per tuple is 200 and the standard deviation is 80. The above imply that each b_N (or c_N) simulation instance has, on average, eight times higher transferring cost per tuple than an a_N simulation instance, while the network heterogeneity of the c_N simulation instances is higher than the network heterogeneity of b_Ns. The response time ratio values ρ'_{b_N} / ρ_{b_N} and ρ'_{c_N} / ρ_{c_N} are also shown in Figure 4 (middle and right bars, respectively). From the above, we can see that as the transferring cost of tuples increases (relatively to their processing cost), the performance of our algorithm is much higher, rendering it more appropriate in environments, where the communication cost dominates. Furthermore, our algorithm is robust to network heterogeneity. For example, the maximum response time deviation is up to 26 times (i.e., the plan produced by Greedy has 26 times higher response time than the optimal one) for the c_N simulation instances (right bars in Figure 4), while the maximum deviation is 11 times for the b_N simulation instances (middle bars Figure 4).

Regarding the efficiency of the proposed algorithm, the mean running time of the optimizer per simulation instance was only 0.3 sec, even for simulation instances of 250 services[3]. This means that our proposal is non computational demanding and can easily run on ordinary machines. More detailed experiments can be found at Tsamoura, Gounaris, & Manolopoulos (2010).

RELATED WORK

Our work relates to the broader areas of distributed query optimization and pipelined operator ordering. Distributed query optimization algorithms differ from their centralized counterparts in that communication cost must be considered and there is a trade-off between total work optimization and the harder problem of response time optimization (Ganguly, Hasan, & Krishnamurthy, 1992). Proposals for the latter case either employ more sophisticated dynamic programming techniques (Kossmann & Stocker, 2000) or resort to heuristics. Response time optimization is largely affected by the types

Figure 4. Response time ratio values between the plans built by the proposed algorithm and the Greedy one presented in (Srivastava, Munagala, Widom, & Motwani, 2006)



of parallelism in the query plan; nevertheless, typically only independent parallelism is investigated in wide area settings (Kossmann, 2000; Deshpande & Hellerstein, 2002), whereas our focus is on pipelined parallelism. There is a lot of work for different settings than the one assumed in this work referencing P2P networks (Huebsch et al., 2005; Ng et al., 2003); however such proposals do not share the same goals and cannot be applied to our problem.

Pipelined operator ordering has been examined for both centralized and distributed environments. In a centralized single-node environment, the problem of minimizing the response time can be optimally solved in polynomial time only if the selectivities are independent (Hellerstein & Stonebraker, 1993; Krishnamurthy et al., 1986); note that if the independence assumption does not hold the problem becomes intractable. In a wide-area environment, the response time optimization problem is transformed to bottleneck cost minimization. In this setting, Srivastava et al. (2006) proposed an algorithm for optimizing select-project-join queries over WSs. However, they assume that the query execution process is simplified through a WSMS which orchestrates data exchange among the services, so that joins can be computed and the heterogeneity of communication costs does not impact on the bottleneck cost metric. As such, our algorithm can be deemed as an extension to Srivastava et al. (2006) for the case when decentralized sequential plans are examined; note that Srivastava et al. (2006) support parallel plans as well, which outperform sequential ones when service selectivities are higher than 1. Braga et al. (2008) deal with a slightly different problem, where IR-style tasks are combined with accurate search tasks in the same query; the goal is to produce a WS invocation plan (either sequential or parallel) in order to obtain the best *k* answers of a query in the presence of access limitations but the algorithm they employ involves exhaustive search of the candidate plans. Deshpande et al. (2005) consider correlated selective attributes but they aim at minimizing the total cost for acquiring the values of the attributes, since they assume that each attribute is assigned an acquisition cost. The plan constructed is a conditional one in the form of a binary decision tree. All these proposals are static. In Babu et al. (2004), the goal is to develop solutions for the ordering of selective operators that are tailored to online, dynamic scenarios. However, the approximate algorithm applies only to the problem of minimizing the total work and assumes selectivities not higher than 1. Complementarily to the above, the algorithm in Sabesan and Risch (2009) tackles the problem of allocating services on host machines with respect to a fixed plan. The algorithm can determine the number of hosts that may execute a WS with a view to minimizing the response time of the submitted query.

None of these works consider the communication costs. Existing solutions for multi-query optimization neglect the communication costs, as well. Liu, Parthasarathy, Ranganathan, and Yang, (2008) and Munagala, Srivastava, and Widom (2007) assume a single-node execution environment, where all operators are selective, potentially correlated and unconstrained. The optimization metric is the minimization of the sum of the operator costs, as in a distributed version of the same problem discussed in Liu, Parthasarathy, Ranganathan, and Yang (2008).

A common characteristic of the proposals mentioned so far is that they build a single plan. For completeness, we mention techniques that define a set of interleaving plans in order to maximize the data flow, which is equivalent to minimizing the bottleneck cost. In Condon et al. (2009), such a tuple routing algorithm is proposed in order to maximize the flow of tuples processed by the filters of the input query. The filters are all selective and unconstrained. The output is a set of serial plans. Each serial plan is assigned a probability weight and when a new tuple enters the system, it is assigned to one of these serial plans with a probability depending on its weight. It must be noted that the flow maximization algorithm considers only the process rates of the filters (i.e., the number of tuples per unit of time) and the potentially heterogeneous communication costs are disregarded. This work is extended in Despande and Hellerstein (2008) to also support proliferative operators and precedence constraints. However, Despande and Hellerstein (2008) is characterized by the limitation of not considering communication costs, too. Note that interleaving plans are not the same as eddies (Avnur & Hellerstein, 2000; Tian & DeWitt, 2003). The former deal with multiple static plans, whereas the latter refer to a single plan that is continuously adapted to changes in the environment.

Finally, a recent work that takes data transmission into account has appeared in Li, Deshpande, and Khuller (2009), which deals with processing of multiple, overlapping, non-parallel queries. The input data is in sources stored on different host machines, while the cost to transfer data between any two hosts varies, as in our problem. Nevertheless, the optimization goal is different; the algorithm in Li et al. (2009) aims to minimize the total cost to transfer data across overlapping queries, whereas we focus on minimizing the response time of a pipelined parallel query.

CONCLUSIONS AND DIRECTIONS FOR FUTURE WORK

In this work, we deal with the optimization of decentralized queries over Web Services. More specifically, we present an algorithm for finding the optimal ordering of pipelined services when the services communicate directly with each other and the communication costs vary. The goal is to minimize query response time, which, due to parallelism, depends on the bottleneck service in the plan. Our algorithm operates regardless of any precedence constraints and selectivity values can be higher than 1. To the best of our knowledge, it is the first attempt to solve this intractable problem. Our algorithm is provably optimal, i.e., always finds the optimal plan, and particularly efficient in terms of running time, as the results of the evaluation reveal. It follows the branch and bound optimization approach and adopts a novel pruning technique in order to reduce the search space. It can yield performance improvements of an order of magnitude in realistic scenarios.

Our work has been motivated by emerging paradigms of distributed data management and can be extended towards several directions in order to fully fulfill modern needs. In the future, we plan to investigate solutions that support more generic plans rather than more simple sequential orderings of operators. In such plans, each service can have multiple inputs and disseminate its results to multiple services simultaneously. The investigation of correlated selectivities and the development of adaptive flavors of the algorithm are also left for future work. Finally, we believe that, in distributed settings, operator ordering solutions must be coupled with resource allocation and scheduling algorithms in order to produce a complete solution. We plan to work to this end in the future, too.

REFERENCES

Alpdemir, M. N., Mukherjee, A., Gounaris, A., Paton, N., Watson, P., Fernandes, A., et al. (2004). Ogsa-dqp: A service for distributed querying on the grid. In *Proceedings of the 9th International Conference on Extending Database Technology* (pp. 858-861).

Avnur, R., & Hellerstein, J. (2000). Eddies: Continuously adaptive query processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (pp. 261-272).

Babu, S., Motwani, R., & Munagala, K. (2004). Adaptive ordering of pipelined stream filters. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (pp. 407-418).

Braga, D., Ceri, S., Daniel, F., & Martinenghi, D. (2008). Optimization of multidomain queries on the web. In *Proceedings of the Very Large Data Base Endowment* (pp. 562-573).

Condon, A., Despande, A., Hellerstein, L., & Wu, N. (2009). Algorithms for distributional and adversarial pipelined filter ordering problems. *ACM Transactions on Algorithms*, *5*(2), 24–34. doi:10.1145/1497290.1497300

Craddock, T., Lord, P., Harwood, C., & Wipat, A. (2006). E-science tools for the genomic scale characterisation of bacterial secreted proteins. In *Proceedings of the All Hands Meeting* (pp. 788-795).

Deshpande, A., Guestrin, C., Hong, W., & Madden, S. (2005). Exploiting correlated attributes in acquisitional query processing. In *Proceedings of the 21st International Conference on Data Engineering* (pp. 143-154).

Deshpande, A., & Hellerstein, J. M. (2002). Decoupled query optimization for federated database systems. In *Proceedings of the 18th International Conference on Data Engineering* (pp. 716-732).

Despande, A., & Hellerstein, L. (2008). Flow algorithms for parallel query optimization. In *Proceedings of the 24th International Conference on Data Engineering* (pp. 754-763).

Ganguly, S., Hasan, W., & Krishnamurthy, R. (1992). Query optimization for parallel execution. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (pp. 9-18).

Gounaris, A., Yfoulis, C., Sakellariou, R., & Dikaiakos, M. (2008). Robust runtime optimization of data transfer queries over web services. In *Proceedings of the 24th International Conference on Data Engineering* (pp. 596-605).

Hellerstein, J. M., & Stonebraker, M. (1993). Predicate migration: Optimizing queries with expensive predicates. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (pp. 267-276).

Huebsch, R., Chun, B. N., Hellerstein, J., Loo, B. T., Maniatis, P., Roscoe, T., et al. (2005).The architecture of pier: An internet-scale query processor. In *Proceedings of the 2nd Conference on Innovative Data Systems Research* (pp. 28-43).

Kossmann, D. (2000). The state of the art in distributed query processing. *ACM Computing Surveys*, *32*(4), 422–469. doi:10.1145/371578.371598

Kossmann, D., & Stocker, K. (2000). Iterative dynamic programming: A new class of query optimization. *ACM Transactions on Database Systems*, *25*(1), 43–82. doi:10.1145/352958.352982

Krishnamurthy, R., Boral, H., & Zaniolo, C. (1986). Optimization of nonrecursive queries. In *Proceedings of the 12th International Conference on Very Large Data Bases* (pp. 128-137).

Li, J., Deshpande, A., & Khuller, S. (2009). Minimizing communication cost in distributed multi-query processing. In *Proceedings of the 25th International Conference on Data Engineering* (pp. 772-783).

Liu, Z., Parthasarathy, S., Ranganathan, A., & Yang, H. (2008). Generic flow algorithm for shared filter ordering problems. In *Proceedings of the 27th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (pp. 79-88).

Liu, Z., Parthasarathy, S., Ranganathan, A., & Yang, H. (2008). Near-optimal algorithms for shared filter evaluation in data stream. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (pp. 133-146).

Malik, T., Szalay, A. S., Budavari, T., & Thakar, A. (2003). Skyquery: A web service approach to federate databases. In *Proceedings of the 1st Conference on Innovative Data Systems Research*

Munagala, K., Srivastava, U., & Widom, J. (2007). Optimization of continuous queries with shared expensive filters. In *Proceedings of 26th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (pp. 215-224).

Ng, W. S., Ooi, B. C., Tan, K. L., & Zhou, A. (2003). Peerdb: A p2p-based system for distributed data sharing. In *Proceedings of the 19th International Conference on Data Engineering* (pp. 633-644).

Nieto-Santisteban, M. A., Gray, J., Szalay, A. S., Annis, J., Thakar, A. R., & O'Mullane, W. (2005). When database systems meet the grid. In *Proceedings of the 2nd Conference on Innovative Data Systems Research* (pp. 154-161).

Sabesan, M., & Risch, T. (2009). Adaptive parallelization of queries over dependent web service calls. In *Proceedings of the 25th International Conference on Data Engineering* (pp. 1725-1732).

Srivastava, U., Munagala, K., Widom, J., & Motwani, R. (2006). Query optimization over web services. In *Proceedings of the 32nd Conference on Very Large Databases* (pp. 355-366).

Tian, F., & DeWitt, D. J. (2003). Tuple routing strategies for distributed eddies. In *Proceedings of the 29th Conference on Very Large Databases* (pp. 333-344).

Tsamoura, E., Gounaris, A., & Manolopoulos, Y. (2010). *Optimal service ordering in decentralized queries over web services*. Retrieved from http://delab.csd.auth.gr/papers/IJKBO10tgm.pdf

ENDNOTES

1 In practice, tuples are transmitted in blocks (Gounaris, Yfoulis, Sakellariou, & Dikaiakos; Srivastava et al., 2006); $t_{i,j}$ is the cost to transmit a block divided by the number of tuples it contains.

2 $t_{N-1,N}$ is always set to 0.

3 For the experiments we have used a machine with a dual core processor, where the CPU clock of each core is at 2.00 GHz and the total memory 2GB.

Efthymia Tsamoura(*) received the Diploma degree in computer science with honors (9.15/10) in 2007. She is currently pursuign the Ph.D. degree in the Computer Science Department, Aristotle University of Thessaloniki (AUTH). During October 2007–June 2008, she was a research assistant in the Multimedia Knowledge Laboratory of the Informatics and Telematics Institute (ITI) of Greece, where she participated in the European project Vidi-Video. Her research interests lie in the field of distributed and adaptive query optimization and grid computing. She was awarded from the State Scholarship Foundation of Greece, for exceptional performance during the academic years 2003-2004, 2004-2005, 2005-2006 and 2006-2007.

Anastasios Gounaris is a lecturer at the Dept. of Informatics of the Aristotle University of Thessaloniki, Greece. Prior to that, he was a visiting lecturer with the University of Cyprus, a research associate with the School of Computer Science of the University of Manchester and he has also collaborated with the Centre of Research and Technology Hellas CERTH. A. Gounaris received his PhD from the University of Manchester (UK) in 2005. He was also awarded an MPhil in Computation in 2002 by UMIST (UK) and a BSc in Electrical and Computer Engineering in 1999 by the Aristotle University of Thessaloniki. His research interests are in the area of Grid and distributed data management, resource scheduling in Grids, autonomic computing, adaptive query processing and semantic technologies. He has served as a program committee member in several international conferences and workshops. He is a member of ACM and IEEE.

Yannis Manolopoulos received his B.Eng (1981) in Electrical Eng. and his Ph.D. (1986) in Computer Eng., both from the Aristotle Univ. of Thessaloniki. Currently, he is Professor at the Department of Informatics of the latter university. He has been with the Department of Computer Science of the Univ. of Toronto, the Department of Computer Science of the Univ. of Maryland at College Park and the Department of Computer Science of the University of Cyprus. He has published more than 200 papers in refereed scientific journals and conference proceedings, whereas he is co-author of four research monographs. His work has received over 2000 citations from over 450 institutional groups. His research interests include Databases, Data mining, Web and Geographical Information Systems, Bibliometrics/Webometrics, Performance evaluation of storage subsystems.