Computing (2012) 94:939–972
DOI 10.1007/s00607-012-0209-9

Optimization of decentralized multi-way join queries over pipelined filtering services

Efthymia Tsamoura · Anastasios Gounaris · Yannis Manolopoulos

Received: 23 March 2012 / Accepted: 3 August 2012 / Published online: 24 August 2012

Abstract The capability to optimize and execute complex queries over multiple remote services (e.g., web services) is of high significance for efficient data management in large scale distributed computing infrastructures, such as those enabled by grid and cloud computing technology. In this work, we investigate the optimization of queries that involve multiple data resources, each of which is processed by non-overlapping sets of remote filtering services. The main novelty of this work is the proposal of optimization algorithms that produce an ordering of calls to services so that the query response time is minimized. The distinctive features of our work lie in the consideration of direct, heterogenous links among services and of multiple data resources. To the best of our knowledge, there is no known algorithm for this problem and the evaluation results show that the proposed algorithms can yield significant performance improvements compared to naive approaches.

Keywords Distributed query processing · Distributed query optimization · Pipelined filter ordering

Mathematics Subject Classification 68M14 (Distributed systems)

1 Introduction

Service-oriented computing (SOC) has been proposed as an efficient approach to developing distributed applications [1]. Among the benefits that emerge from SOC

E. Tsamoura · A. Gounaris · Y. Manolopoulos (✉)
Aristotle University of Thessaloniki, Thessaloniki, Greece
e-mail: manolopo@csd.auth.gr

E. Tsamoura
e-mail: etsamour@csd.auth.gr

A. Gounaris
e-mail: gounaria@csd.auth.gr

are the easy and rapid development of evolvable, platform-independent and arbitrarily complex distributed applications. Seeking to benefit from the opportunities that arise from SOC, the web and grid data management infrastructures are moving towards a service-oriented architecture by putting their data and analysis tools behind services. As a consequence, there is a growing interest in systems that are capable of processing complex queries (i.e., tasks) utilizing remotely deployed services. Examples of such systems are scientific and business workflow management systems (e.g., Taverna [2], Triana [3], Pegasus [4]), service-based execution engines (e.g., OGSA-DAI/DQP [5, 6], YQL[1], BPELSE[2], Apache ServiceMix[3]) and data mashup platforms (e.g., Yahoo! Pipes[4], IBM Mashup Center[5], WSO2 Mashup Server[6]). These families of systems focus on different aspects. For example, the workflow management systems focus mostly on issues such as efficient dispatching of independent tasks to nodes provided that the logical order of calls to services has been fixed during workflow specification (e.g., Pegasus), while the data mashup platforms focus on the development of tools that facilitate the combination of different services and data resources (e.g., Yahoo! Pipes).

In the vast majority of the systems, the users manually specify the invocation order of services. However, modifying the service execution order, without affecting the correctness of the query result, may lead to significant performance improvements. For example, it may be more beneficial to place a highly selective service at the beginning of the service workflow, since, the rest of the services may process less data and thus the workflow execution time may be reduced. An example of a bio-informatics query, where changes in the order of service invocations lead to performance improvements, is presented in [7]. Typically, finding a feasible service invocation plan is not a trivial task, especially in a wide-area environment [8]. This is due to the heterogeneity of the service characteristics (e.g., data processing costs and selectivity) and the heterogeneity of the underlying communication network. The above discussion brings up an important optimization problem: to find the order of invocation of the remote services in a query that minimizes the query execution time.

An instance of the above problem is in the context of a wide-area environment that employs pipelined parallelism. Pipelined parallelism allows data already processed by a service to be processed by the subsequent service in the query plan at the same time as the former service processes new data items. In such a setting, the query execution time equals the time spent by the slowest (bottleneck) service to process the incoming data and to send the results to a subsequent service [9]. The related work section provides an overview of the works recently proposed in the literature that deal with several variants of the pipelined service ordering problem.

[1] http://developer.yahoo.com/yql/.

[2] http://wiki.openesb.java.net/wiki.jsp?page=bpelse.

[3] http://servicemix.apache.org/home.html.

[4] http://pipes.yahoo.com/pipes/.

[5] http://www-01.ibm.com/software/info/mashupcenter/.

[6] http://wso2.com/products/mashup-server/.

1.1 Context of this work

In this work, we deal with the problem of finding the ordering that minimizes the response time of pipelined multiway join queries over remote filtering services that process data from multiple input resources in a decentralized fashion. In the sequel, we briefly describe the motivation behind and the novelty related to this problem. The motivation to deal with multiway join queries stems from the fact that a plethora of applications from different areas combine data from multiple resources [10,11]. For example, an application that performs meteorological studies may have to combine data from different sensor deployments, where each sensor deployment performs different types of measurements. Other examples are met in bioinformatics and distributed video surveillance applications. A biologist may have to perform comparisons between proteins from different biological resources, while a distributed video surveillance application may have to find spatial regions that are captured by multiple video cameras concurrently.

Furthermore, many of the operations that are applied on the input data perform some kind of filtering [11,12]. Continuing with the aforementioned examples, a meteorological application may be interested in studying only a portion of the sensed measurements, say, those that exceed a certain threshold. A biological application, in turn, may consider only those proteins from a protein database that contain a specific DNA sequence, while a video surveillance application may need to analyze only the video segments from a video database where an abnormal event has occurred. Filtering may be also performed by look-up services. A look-up service takes one or more data values as input and performs key-foreign key joins with a base data source, i.e., it checks if the input data values are present in a base data source or not. In the former case, the service returns the matched data along with one or more attribute values associated with the input data; otherwise it prunes the input data. A short example is given below. A bioinformatics service may receive as input a protein id and return the corresponding protein sequence if the id is present. This service is filtering because not all protein ids may be present in the database, which database is wrapped as a service. Examples of look-up services are given in Sect. 2.

The different kinds of applications where the problem of multiway join query optimization over remote filtering services is met has driven us to take different assumptions regarding the execution environment. More specifically, we address two variants of the aforementioned problem. In the first variant, the services are both logically and physically clustered, while in the second variant, services are only logically clustered. By logical clustering we mean that different, non-overlapping sets of services process the data of each input resource. By physical clustering we mean that each set of services that process the data from a specific data source is allocated to hosts that extend over distinct geographical areas. Examples where services are physically clustered are met in modern sensor data processing applications [13,14].

A distinctive feature of this work is our assumption that the input data is processed in a decentralized fashion meaning that services communicate directly with each other. Furthermore the communication links among the services are heterogeneous meaning that different service pairs communicate (i.e., exchange data) with different network costs. This aspect constitutes a significant difference with similar proposals, such

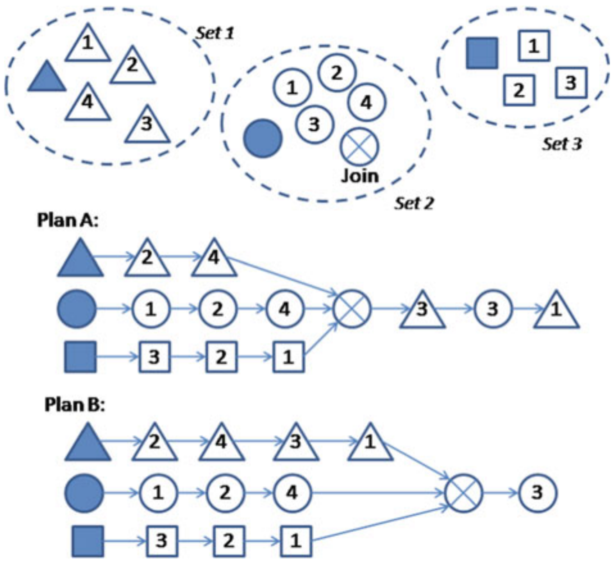


Fig. 1 Example plans of a 3-input join query over services

as [9,15,16], which either overlook communication links or consider them as being homogeneous.

Figure 1 presents an example of the problem that we consider. It refers to a query over three distinct inputs. Each of these inputs, apart from being joined with the other two, is processed by a distinct set of filtering services (triangular, circular and square shapes in the figure). Filtering services may drop some of the tuples they process. The data inputs can be represented as services, too, and are depicted as filled shapes in Fig. 1. One of these service sets contains a multi-way join service, shown as $\otimes$. It is out of our scope to investigate the specific implementation of pipelined multi-way joins, since efficient proposals already exist (e.g., [17]). Let us assume that there are no precedence constraints among the services, apart from the obvious constraint that the input services must be placed at the beginning. All services are deployed in predefined hosts, and the objective is to minimize the query response time. There are multiple orderings of services that produce the result set. Plans A and B in the figure are two examples that differ in the order they call the services. The aim of this work is to find the optimal one. Section 2 describes two motivation examples in more detail, whereas Sect. 3 formally presents the problem we deal with.

1.2 Contributions

In this work, we propose algorithms for multiway join query optimization over filtering services that are either logically or both logically and physically clustered. Both algorithms adopt a two-phase optimization approach. In the first phase, a preliminary service ordering is found, while the second phase heuristically tries to improve the previously found service plan. The approach that is adopted in the first phase is to create, for each input data source, a linear ordering utilizing only the services that are selected to process the data of that data source. The output data of each linear service ordering is then fed to the multiway join service. This approach allows us to investigate each input separately, thus reducing the problem of optimizing multi-join queries to that of optimizing single-input queries. For the latter, we employ the recently proposed efficient algorithm in [18], i.e., our previous work in [18] serves as a building block in order to develop algorithms for multiple inputs. During the second phase, one or more services are removed from the linear service orderings and placed after the multiway join service. We theoretically prove the performance of the proposed two-phase optimization algorithms. In particular, we investigate under which conditions the two-phase optimization approach generates provably optimal multiway join plans. We also theoretically prove when the second phase can lead to further improvements or not.

To the best of our knowledge, this is the first attempt to solve this problem. Note that our solution is independent of any specific standardization regarding service communication, data access, orchestration and choreography. The most relevant work with ours is the one presented in [9], which however, does not consider queries over multiple data sources and does not assume that services communicate with each other directly.

In summary the contributions of this work are:

- one algorithm, called 2P-SO, for the problem of multiway join query optimization over services that are both logically and physically clustered
- an extension of 2P-SO, called A2P-SO, for the more general problem where services are only logically clustered
- a theoretical analysis on the performance of the proposed algorithms
- an experimental analysis of the benefits of the proposed algorithms compared to other approaches.

The remainder of the paper is structured as follows. Section 2 presents two motivating examples. Section 3 formalizes the problem. Section 4 discusses our solution when the communication cost is the dominant cost and the services are physically clustered. In Sect. 5, we relax these assumptions, whereas the evaluation is in Sect. 6. Section 7 deals with the related work and Sect. 8 concludes the paper.

2 Motivating examples

In this section we will present two examples of real-life applications which have motivated our work.

Example 1 Sensor networks have gained significant attention the last decades due to the importance and impact of applications such as environmental and healthcare monitoring.

Suppose that there exist two sensor deployments that cover a specific area each one of them owned by a different scientific laboratory. The first deployment aims at sensing the relative humidity, while the second deployment aims at sensing the soil moisture. Both laboratories employ an infrastructure in order to archive and analyze the sensed data, while they also provide several (web) services in order to publicly share sensor measurements, which are deployed on hosts owned by each laboratory's cluster. The services of the first laboratory are tagged with $\triangle$, while the services of the second laboratory are tagged with $\square$. The sensor measurement database of the first laboratory is wrapped as a service, denoted by $S_1^{\triangle}$. Parameterizing $S_1^{\triangle}$ with a timeperiod $[\tau_l, \tau_u]$, we get all humidity measurements taken during that timeperiod. Each humidity measurement hd is associated with (i) the identifier (sid) of the sensor that performed that measurement, (ii) the timestamp (τ) when this measurement is taken and (iii) a unique measurement identifier (key). Service $S_2^{\triangle}$, in turn, returns for an input sensor identifier the location of that sensor.

Sensors also periodically report at a base station their operational characteristics regarding their energy reserves. It is important to know the runtime levels of a sensors' energy source as they heavily affect the quality of the performed measurements. Given a sensor identifier sid and a timepoint τ (e.g., the timestamp associated with a perform humidity measurement), service $S_3^{\triangle}$ returns the energy reserves $energy$ at time τ of the sensor with identifier sid. As the sensors do not report their runtime energy reserves with the same frequency they take humidity measurements, service $S_3^{\triangle}$ returns for an input timepoint the energy reserves that are reported as close as possible to that timepoint. Service $S_3^{\triangle}$ can be also parameterized with an energy threshold θ_{energy}, so as to return a sensor's energy reserves only if they exceed the user defined threshold. The services of the second sensor deployment have similar functionality.

Services $S_2^{\triangle}$ and $S_3^{\triangle}$ are look-up services, whereas $S_3^{\triangle}$ is also a filtering service. The selectivity of $S_2^{\triangle}$ is always 1, since each sensor is deployed on a unique location, while the selectivity of $S_3^{\triangle}$ is in general below 1: service $S_3^{\triangle}$, given a sensor identifier sid and a timepoint τ, it first looks for the energy reserves of sensor sid at timepoint τ and then checks if the reported reserves are above a user defined threshold. In case of success, the service appends to the input data the associated energy reserves. Otherwise, service $S_3^{\triangle}$ prunes the input tuple.

In both sources, the location is represented by the geographical coordinates of a bounding rectangle. However, the rectangles are not the same in the two deployments, i.e., the deployments refer to different sets of rectangles. In order to facilitate the integration, someone has to use a third service that answers queries about the overlapping of such rectangles. Service $S_{join}^{\otimes}$ is developed by an end user who wants to process the measurements and returns a number between 0 and 1 describing the degree of overlapping between the rectangles given two input locations.

Suppose that someone wants to answer the following query: "find the relative humidity and soil moisture at locations with degree of similarity 0.5 within a timeperiod $[\tau_l, \tau_u]$. The energy reserves of the sensors the timepoints when the measurements were performed must be above θ_{energy}". There are several possible service invocations in order to answer this query. However, there exist a series of precedence constraints regarding the allowable positions of services $S_1^{\triangle}$, $S_1^{\square}$ and $S_2^{\triangle}$, $S_2^{\square}$ in the resulting

$$\begin{tikzcd}
S_1^{\triangle} \arrow[r] & S_2^{\triangle} \arrow[dr] & & & \\
& & S_{join}^{\otimes} \arrow[r] & S_3^{\triangle} \arrow[r] & S_3^{\square} \\
S_1^{\square} \arrow[r] & S_2^{\square} \arrow[ur] & & &
\end{tikzcd}$$

(a)

$$\begin{tikzcd}
S_1^{\triangle} \arrow[r] & S_2^{\triangle} \arrow[r] & S_3^{\triangle} \arrow[dr] & \\
& & & S_{join}^{\otimes} \\
S_1^{\square} \arrow[r] & S_1^{\square} \arrow[r] & S_3^{\square} \arrow[ur] &
\end{tikzcd}$$

(b)

Fig. 2 Two alternative plans for answering the query "find the relative humidity and soil moisture at locations with degree of similarity 0.5 within a timeperiod $[\tau_l, \tau_u]$. The energy reserves of the sensors the timepoints when the measurements were performed must be above θ_{energy}" using the services described in Example 1. Filtering out the measurements taken by sensors with low energy reserves is done **a** after and **b** prior to finding the locations that overlap significantly enough through $S_{join}^{\otimes}$

plan. Services $S_1^{\triangle}$ and $S_1^{\square}$ must be invoked before any other service as they provide access to the stored measurements, while services $S_2^{\triangle}$ and $S_2^{\square}$ must be invoked before $S_{join}^{\otimes}$ as the latter operates on the bounding rectangles of the geographical coordinates of the sensors. The evaluation strategy followed by the plan shown in Fig. 2a first retrieves the measurements collected by the two deployments within this time period. The returned measurements are then used to obtain the locations where these measurements have been collected from. The locations are sent to service $S_{join}^{\otimes}$ which calculates the similarity degree for the locations of each pair of measurements and selects those measurement pairs that satisfy the similarity predicate. Finally, for each of the returned measurements, services $S_3^{\triangle}$ and $S_3^{\square}$ check if the sensor's energy reserves associated with those measurements are above the user defined threshold θ_{energy}. An alternative approach is adopted by the plan shown in Fig. 2b, where the filtering of measurements taken by sensors with low relative reserves is done prior to finding the locations that overlap significantly enough.

From Example 1, we can see that the services have the following characteristics: (i) they are selective as they filter out portion of the stored data based on some criteria and (ii) they are both physically and logically clustered (a specific set of services has to process the data of each sensor deployment, while these services are deployed on each laboratory's cluster).

Example 2 The second example is a generalization of Example 1. Suppose that we must perform advanced analysis operations on the sensor measurements supplied by the two aforementioned scientific laboratories. The scientific laboratories, however, provide only services that can access the archived data, while the rest of the operations that must be applied to the sensor measurements are provided through web services by, say, three other parties that belong to different organizational domains. In contrast to Example 1, where the services that had to be applied to the input data items were physically clustered, the services that had to be applied to the measurements are only logically clustered, e.g., the sensor measurements of one laboratory may be processed by services belonging to different organizational domains that are physically located at arbitrary places.

Table 1 The notation used throughout the paper

Symbols	Description
L	Number of input data resources
X^j	Data from the j-th data resource, $j = 1, \ldots, L$
$\mathcal{W}^j$	The set of services processing X^j
S_i^j	The i-th service of $\mathcal{W}^j$, where $\mathcal{W}^j$ does not contain the multi-way join service
S_{join}^k	The multi-way join service, which belongs to $\mathcal{W}^k$
c_i^j	The average cost of service S_i^j per input tuple
σ_i^j	The selectivity of service S_i^j
$\mathbf{c}_{join}^k = [c_1, \ldots, c_L]$	A vector containing the average processing costs of S_{join}^k per input tuple for each X^j
$\boldsymbol{\sigma}_{join}^k = [\sigma_1, \ldots, \sigma_L]$	A vector containing the selectivity S_{join}^k per X^j
$t_{i,r}^{j,l}$	The average cost to transfer a tuple from S_i^j to S_r^l
$T_{i,r}^{j,l}$	The aggregate cost of the S_i^j with respect to S_r^l, where $T_{i,r}^{j,l} = c_i^j + \sigma_i^j t_{i,r}^{j,l}$
$\mathcal{L}^j$	Linear ordering of services from $\mathcal{W}^j$ placed before S_{join}^j
ρ^j	The bottleneck cost of the $\mathcal{W}^j$ subplan
$\mathcal{C}$	Linear ordering of services after S_{join}^j
$\mathcal{P}$	The complete multi-way join plan
$R_i^j(P)$	The product of selectivities of all services preceding $\mathcal{S}_i^j$ in plan $\mathcal{P}$

3 Problem formulation and background

Consider L data resources. Each data resource produces a data stream X^j, $j = 1 \ldots, L$ of finite length. The queries we examine define that (i) the data from each resource should be joined together, e.g., using a condition on common attributes and perhaps a time window if the data streams are time-stamped, and (ii) the data from each resource should be processed by several filtering and look-up services and the processing cost of each tuple in X^j corresponds to a service call. Let $\mathcal{W}^j$ be the set of services that process data from X^j. S_i^j denotes the i-th service of $\mathcal{W}^j$ set. One of the sets, lets say $\mathcal{W}^k$, contains a specific join service, denoted by S_{join}^k, which implements the multi-way join operation. For readability reasons, the notation is summarized in Table 1.

The average cost of $S_i^j \in \mathcal{W}_j$ to process an input tuple is c_i^j. For the data resource services, this cost may correspond to the cost of producing tuples, which is the inverse of the tuple production rate. For the join service, the average processing cost may differ for tuples originated from different resources; as such, there is a different cost for each of the L inputs. Let $t_{i,r}^{j,l}$ be the average cost to transfer a tuple from the i-th service in $\mathcal{W}^j$ to the r-th service in $\mathcal{W}^l$. Note that, in practice, tuples are transmitted in blocks [9]; in that case, $t_{i,r}^{j,l}$ is the cost to transmit a block divided by the number of tuples it contains. Finally, the selectivity of S_i^j, which defines the average number of output tuples per input tuple, is denoted by σ_i^j. For the join service, there is a different selectivity value for each input. We assume that all services are selective, i.e., $\sigma_i^j \leq 1$[7], apart from the join selectivities that are higher than 1, as is the case in most real queries. Look-up services that fall into this category include services that in practice perform key-foreign key joins and potentially apply predicates, e.g., for a given location id, retrieve its bounding rectangle if it is in the north hemisphere.

Although our techniques can be applied to both push and pull models of inter-service communication, we will present only the push case, where the sender initiates the data communication process. The aggregate per-tuple cost of the sender service S_i^j with respect to the receiver service S_r^l is then $T_{i,r}^{j,l} = c_i^j + \sigma_i^j t_{i,r}^{j,l}$, which takes into account the processing cost of the input and the transmission cost of the output. The previous formula shows that a consequence of considering heterogeneous links is that the cost of a service also depends on the next service in the plan. This is a major difference from previous work where the cost incurred by a service S_i^j was only affected by the services that preceded S_i^j in the plan (e.g., [9,15]).

Costs and selectivities are assumed to be independent of each other. For example, the selectivity of a filtering or the join service does not depend on the selectivities of the preceding services.

Our goal is to build a plan $\mathcal{P}$ (i.e., to define the appropriate ordering of the services) that has the minimum response time. As shown from Fig. 1, in the generic case, a plan is composed of L linear orderings $\mathcal{L}^j$ before the join service, and an additional ordering $\mathcal{C}$ after the join service[8]. Each $\mathcal{L}^j$ comprises services only from $\mathcal{W}^j$, whereas $\mathcal{C}$ may contain services from any set. In a linear ordering, each service, except the join one, receives input from a single service and sends output tuples to up to one service.

In general, precedence constraints exist between filtering services. The more precedence constraints exist, the less service permutations the optimizer has to investigate. Precedence constraints can exist only between services from the same set, with the exception of the join service. $S_r^j \prec S_i^j$ means that S_r^j must appear before S_i^j. Precedence constraints apply to all data resource services to state that they must precede all other service calls.

Because of the pipelined model of execution, according to which the services process tuples concurrently, the query response time can be safely approximated by the time the slowest service requires to complete its work. For this reason, the problem objective can be expressed as the minimization of the bottleneck cost. Suppose that S_i^j is placed immediately before S_r^l in plan $\mathcal{P}$. Then, for each call, S_i^j spends $T_{i,r}^{j,l}$ time units. Also, the proportion of X^j tuples that reaches S_i^j is the product of the selectivities of all services preceding S_i^j up to the data resource service of X^j, $R_i^j(\mathcal{P})$. Thus the response time for S_i^j in $\mathcal{P}$ is the product of the number of tuples that arrive at S_i^j (i.e., $||X^j|| \cdot R_i^j(\mathcal{P})$) and of the cost spent by S_i^j to process this data and send

[7] Note that throughout the rest of the work, the terms filters and services are used interchangeably.

[8] Note that we restrict our search space to single linear orderings per data resource; in [9] multiple parallel such orderings are examined, which is something we plan to explore in the future.

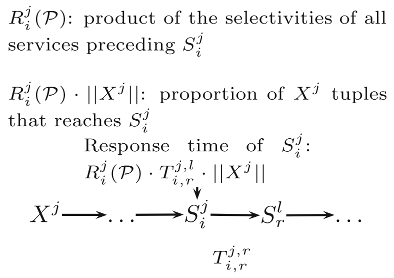

$R_i^j(\mathcal{P})$: product of the selectivities of all services preceding S_i^j
$R_i^j(\mathcal{P}) \cdot ||X^j||$: proportion of X^j tuples that reaches S_i^j
Response time of S_i^j: $R_i^j(\mathcal{P}) \cdot T_{i,r}^{j,l} \cdot ||X^j||$
$X^j \longrightarrow \ldots \longrightarrow S_i^j \longrightarrow S_r^l \longrightarrow \ldots$
$T_{i,r}^{j,r}$

Fig. 3 The response time of S_i^j is calculated by the product of (i) the fraction of tuples in X^j that are not filtered out by any of the services that precede S_i^j in plan $\mathcal{P}$ (i.e., $R_i^j \cdot ||X^j||$) and (ii) the cost spent to process and send the output tuples to S_r^j (i.e., $T_{i,r}^{j,r}$)

the results to the subsequent service S_r^l (i.e., $T_{i,r}^{j,l}$), i.e.,

$$cost(S_i^j) = R_i^j(\mathcal{P}) \cdot T_{i,r}^{j,l} \cdot \parallel X^j \parallel$$

where $\parallel X^j \parallel$ is the size of X^j. A graphical interpretation of the equation is shown in Fig. 3. For the join service, if we assume that every input is processed by a separate parallel thread, the average processing cost is computed for each input separately, and the total processing cost of the join, is the maximum of the processing cost for each input. Although the processing cost incurred by the join service may be much higher than the ones incurred by the filtering services, we assume that it does not comprise the bottleneck of the whole plan, since in a wide-area distributed environment the data transferring cost typically dominates the processing one. In order to compute the size of the join output, it is adequate to examine the selectivity with regards to a single, arbitrary input only. More details about how multi-way joins may be implemented can be found at [17], where it is explained that multiway joins implemented as a single operator outperform trees of binary joins.

Based on the above, our problem can be formulated as a min–max problem, where we try to build a plan $\mathcal{P}$ so that the maximum response time of an individual service is minimized.

Note that, if all $\parallel X^j \parallel$ values are equal, they can be factored out, and we can consider only per tuple service costs. The per tuple cost of each S_i^j followed by S_k^l can be estimated by the simpler formula $cost(S_i^j) = R_i^j(\mathcal{P}) \cdot T_{i,r}^{j,l}$. By tweaking the selectivities of services corresponding to data resources, it is straightforward to render all $\parallel X^j \parallel$ values equal for simplicity reasons. Also, minimizing the query response time makes sense only when the data resources do not produce tuples infinitely; however, we can still apply the main concept of our methodology to continuous streams by considering the rate of result generation, which is the inverse of the per tuple query response time. Finally, if the results must be delivered to a specific node, it is convenient to introduce a result gathering service in the query, which must be placed after all other services in the query plan. If the communication costs between the result gathering site and all other sites are equal, then we can safely ignore the result gathering phase during query optimization. In this work, we do not deal with result gathering, explicitly.

The proposed algorithms are based on detailed statistics about the per-tuple processing costs of services, the per-tuple data transferring costs and the services' selectivity. In this work we assume that these statistics are known and do not change frequently and that the metadata regarding the location and the binding patterns of the services do not vary rapidly. There exist several techniques to acquire the necessary information. For example, we can employ a profiling technique such as the one described in [16] to obtain an estimate of the service processing costs and selectivity. Furthermore, knowing the physical locations of the hosts where the services are deployed, we can employ a Network Coordinates (NC) technique (e.g., [19]) in order to approximate the network delay to transfer data between services by the network delay incurred to transfer data between the hosts where the corresponding services are deployed to. Other work regarding web service profiling can be found in [20].

4 Optimization for physically clustered services

In this section, we present an algorithm, called *2 Phase-Service Ordering* (2P-SO), for building multi-way join plans, when the services are not only logically, but physically clustered as well, i.e., the set of services $\mathcal{W}^j$ is allocated to hosts that extend over distinct geographical areas. Furthermore, it is assumed that the processing cost is negligible relatively to the communication cost of data. The above assumptions imply that

$$T_{i,r}^{j,j} \leq T_{i,r}^{j,l} \tag{1}$$

The meaning of the condition is that the aggregate cost of a service with regards to another service in the same cluster never exceeds the aggregate cost when the latter service belongs to a different cluster.

The algorithm consists of two phases. During the first phase, a multi-way join plan is built, while the second phase aims to refine the bottleneck cost of the initial plan by removing services (one service at a time) from the subplans that precede the multi-way join service. The second phase is optional, since, under certain conditions, it can be safely omitted. In particular, it is proved through several theorems that the employment of the second phase cannot produce a new plan with lower bottleneck cost under certain cases.

An experimental evaluation of 2P-SO shows that the algorithm reaches to optimal multiway join plans quite often under the above assumptions. The same rationale, i.e., the creation of a preliminary multiway join plan and its iterative refinement, can be also applied when the services are logically clustered but Eq. (1) does not hold. However, in Sect. 5 we discuss about a generalization of the above rationale. The section starts with a description of the rationale behind 2P-SO, continues with a description of the two phases of 2P-SO and ends with the analysis of the cases, where the second phase is omitted.

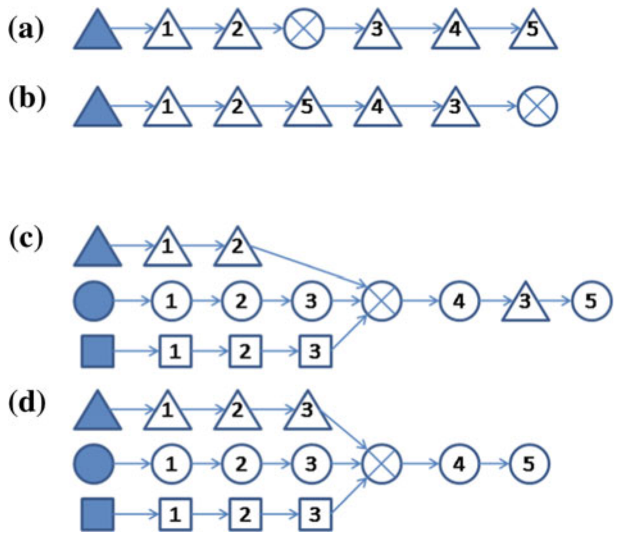


Fig. 4 Let $W^{\triangle}$, $W^{\square}$ and $W^{\bigcirc}$ be three sets of services, where $W^{\bigcirc}$ contains the multiway join service $S^{\bigcirc}_{join}$. Examples of plans that, despite the fact that they are semantically equivalent [plan a (c) is equivalent to plan b (d)], they have different response times

4.1 Rationale of the proposed algorithm

Two observations constitute the rationale behind 2P-SO. Consider a single-input linear ordering, as shown in Fig. 4a. This single-input ordering may be a part of a multi-input join plan. $\mathcal{L}^{\triangle}$ comprises the data resource service and the services numbered 1 and 2, whereas services 3,4 and 5 belong to $\mathcal{C}$. For the join service, since we investigate only a specific data resource, we omit the rest of the inputs.

We transform such a plan in a plan where $\mathcal{L}^{\triangle}$ includes all services from $\mathcal{W}^{\triangle}$. This is performed by removing the services after the join service $S^{\bigcirc}_{join}$ and appending them just before the join in reverse order (see Fig. 4b). The second plan is preferable in terms of lower bottleneck cost. The reason is that $T^{\triangle,\bigcirc}_{2,join} \geq \{T^{\triangle,\triangle}_{2,5}, T^{\triangle,\triangle}_{5,4}, T^{\triangle,\triangle}_{4,3}\}$, since the join service belongs to a different cluster. Also, if computation costs are negligible and because (i) the join selectivity is higher than the other selectivities and (ii) $t^{\bigcirc,\triangle}_{join,3} \approx t^{\triangle,\bigcirc}_{3,join}$, we can argue that $T^{\bigcirc,\triangle}_{join,3} \geq T^{\triangle,\bigcirc}_{3,join}$. Taking into account that the services are selective, so that $R^{\triangle}_{3}(Plan(b)) \leq R^{O}_{join}(Plan(a))$ holds as well, the bottleneck cost of plan (a) in Fig. 4 is never lower than that of plan (b), regardless of its exact positioning in the plan.

The second observation refers to the bottleneck cost of $\mathcal{C}$. Let us assume that, initially, $\mathcal{C}$ contains a service that does not belong to the same group with the join service, as it is the case with the triangular service (△) tagged with number "3" (see Fig. 4c). If this service is moved to the end of the corresponding subplan before the join service (see Fig. 4d), then the bottleneck cost of $\mathcal{C}$ cannot increase. This is because (i) no service in $\mathcal{C}$ in the new plan processes more data given that services, except join, are selective and (ii) $\mathcal{C}$ does not involve transmission costs between services from different clusters any more.

4.2 Algorithm description

Based on the two aforementioned observations, 2P-SO tries to build plans, where $\mathcal{C}$ is either empty or contains services solely from the set of the join service. This means that there are no parts of a multi-join plan where the services are mixed, and as such, each join input can be treated separately. In other words, the problem of devising the optimal plan for a multi-input query can be reduced to the same problem for a single input. This is performed in the first phase of 2P-SO. We also add a second phase, to cover cases, where the first phase can be further improved. The two phases of the proposed algorithm are elaborated below. From now on and throughout the end of this section, $\mathcal{W}^k$ denotes the set that contains the multi-way join service.

Phase 1 For each input, we build an optimal linear ordering $\mathcal{L}^j$, each of which comprises all services in $\mathcal{W}^j$. During the linear plan construction procedure, both the services in $\mathcal{W}^j$ and the join service S^k_{join} are taken into account, even if $j \neq k$. If $j \neq k$, the join service is always placed at the end of the plan as discussed above. This can be also enforced by adding the precedence constraint $S^j_i \prec S^k_{join}$. Although its positioning is fixed, it is considered during optimization because it may affect the orderings of the rest services; this is due to the fact that the communication costs between services in $\mathcal{W}^j$ and the join service S^k_{join} may vary[9]. If $j = k$, the join service may be placed anywhere. The part of the ordering of $\mathcal{W}^k$ before S^k_{join} corresponds to $\mathcal{L}^k$, and the part after the join service forms $\mathcal{C}$ in the complete plan. The complete multi-way join plan is formed by connecting the output of the last service of every $\mathcal{L}^j$ with one of the inputs of the multi-way join service. Phase 1 outputs plans similar to the ones shown in Fig. 4d.

Phase 2 If the bottleneck service of the complete plan appears in any of the $\mathcal{L}^j$ subplans, then we check whether moving that filtering service at the end of the subplan $\mathcal{C}$ can improve the overall query response time. In that case, we rebuild the corresponding $\mathcal{L}^j$ without including the bottleneck service. The intuition is that, if the bottleneck filtering service is at the end of the overall plan, it will process fewer data tuples. This phase is repeated until it cannot yield performance improvements.

The core building block of 2P-SO is a solution for the optimal single-input service ordering. Such a solution has been proposed in [18] and is provably optimal. Interestingly, any other algorithm for linear service ordering can be used instead. A brief description of the algorithm is given in Sect. 5.

The steps of 2P-SO are shown in Fig. 5.

[9] The processing cost and the selectivity of S^k_{join} when building $\mathcal{L}^j$ are given by $\mathbf{c}^k_{join}(j)$ and $\boldsymbol{\sigma}^k_{join}(j)$, respectively.

```
Inputs
  $\mathcal{W}^j, j = 1, \ldots, L$: The set of services that process the data of $X^j$.
  The algorithm proposed in [18] for building linear service plans.
Outputs
  A multi-way join plan $\mathcal{P}$.

1: $exit \leftarrow$ false;
2: {Phase 1}
3: Let $\mathcal{W}^k$ be the set of services that contains the multiway join service $S^k_{join}$;
4: $\mathcal{C} \leftarrow \emptyset$; {$\mathcal{C}$ is the linear subplan of $\mathcal{P}$ consisting of services placed after the multiway join service $S^k_{join}$.}
5: for $j = 1, \ldots, L$ do
6:   if $j \neq k$, i.e., the service set $\mathcal{W}^j$ does not contain the multiway join service $S^k_{join}$ then
7:     Build a linear service plan $\mathcal{L}^j$ utilizing both the services in $\mathcal{W}^j$ and $S^k_{join}$ imposing the constraint that $S^k_{join}$ is always placed after any service in $\mathcal{W}^j$ in the resulting plan, i.e., $S^j_i \prec S^k_{join}$;
8:     Let $\mathcal{L}^j$ be the resulting linear service plan without the multiway join service $S^k_{join}$;
9:   else
10:    Build a linear service plan utilizing the services in $\mathcal{W}^k$ without imposing any additional precedence constraint;
11:    Let $\mathcal{L}^k$ be the part of the ordering of $\mathcal{W}^k$ before $S^k_{join}$;
12:    if one or more services of $\mathcal{W}^k$ are placed after $S^k_{join}$ in the output linear plan then
13:      $\mathcal{C}$ is assigned to the part of the ordering of $\mathcal{W}^k$ after the multiway join service;
14:    end if
15:  end if
16: end for
17: Create a multi-way join plan $\mathcal{P}$ using the subplans $\mathcal{L}^j, j = 1, \ldots, L$ and $\mathcal{C}$ by connecting the output of the last service of every $\mathcal{L}^j$ with one of the inputs of the multi-way join service;
18: {Phase 2}
19: while $!exit$ do
20:   if the bottleneck service of $\mathcal{P}$ appears in any of the $\mathcal{L}^j$ subplans and Theorems 1 to 3 in Section 4.3 are not satisfied then
21:     Let $S^j_{bottleneck}$ be the bottleneck service of $\mathcal{L}^j$;
22:     Rebuild the corresponding subplan $\mathcal{L}^j$ without including $S^j_{bottleneck}$; {The rationale for building the new subplan is similar to the one presented in lines 5-10 of the algorithm.}
23:     Rebuild the subplan $\mathcal{C}$ utilizing the services already added to it and $S^j_{bottleneck}$ imposing the constraint that the join service is always placed before any other service already added to $\mathcal{C}$;
24:     Create a new multiway join plan $\mathcal{P}'$ by updating the subplans $\mathcal{L}^j$ and $\mathcal{C}$ in $\mathcal{P}$ with the newly created ones;
25:     if the cost of $\mathcal{P}$ is higher than the cost of $\mathcal{P}'$ then
26:       $\mathcal{P} \leftarrow \mathcal{P}'$;
27:     else
28:       $exit \leftarrow$ true;
29:     end if
30:   else
31:     $exit \leftarrow$ true;
32:   end if
33: end while
```

Fig. 5 The steps of 2P-SO

4.3 On the effectiveness of the second phase of 2P-SO

After presenting the basic steps of 2P-SO, we continue with a description of the cases, where the second phase of the algorithm can be safely omitted because it is not capable of improving on the output of the first phase.

Three cases exist, under which, the second phase is not necessary to be triggered. In the first case, it is proven that the algorithm builds the optimal multi-way join plan during the first phase, while in the other two cases, it is proven that the second phase cannot lead to further performance improvements.

The first case is presented in Theorem 1. Theorem 1 proves that when the bottleneck subplan $\mathcal{L}^j$, i.e., the subplan where the maximum cost is incurred, has the minimum cost among the subplans built from any subset of the services in $\mathcal{W}^j$, then the initially found multi-way join plan is optimal. The bottleneck cost of a linear plan cannot be lower than the cost of the least expensive pair of services placed at the beginning of the linear plan [18]. Thus, in order to check if a linear subplan $\mathcal{L}^j$ incurs the minimum cost among the feasible plans that can be built from any subset of services in $\mathcal{W}^j$, $O(|\mathcal{W}^j|^2)$ service pairs need to be checked. If there exists a source service, as in our case, the number of service pairs that are checked is reduced to $O(|\mathcal{W}^j|)$. Theorem 1 holds either for the case where Eq. (1) is satisfied or not, however, all services, apart from the join service, must be selective.

Theorem 1 *When the bottleneck subplan $\mathcal{L}^j$ has the minimum cost among the subplans built from any subset of the services in $\mathcal{W}^j$, then the multi-way join plan that is found during the first phase of 2P-SO is optimal.*

Proof In order to prove the above theorem, it suffices to prove that if any subset of services in $\mathcal{L}^j$ is moved after the join service, then the new multi-way plan $\mathcal{P}'$ incurs cost not lower than before.

Let ρ^j be the bottleneck cost of $\mathcal{L}^j$. Since $\mathcal{L}^j$ incurs the maximum cost among the rest of the subplans that precede the join service and $\mathcal{C}$, then the bottleneck cost of the produced multi-way join plan is $\rho = \rho^j$. Let $\mathcal{P}'$ be another multi-way join plan with bottleneck cost ρ', where the services in $\mathcal{W}^j$ form another linear subplan $\mathcal{L}^{j'}$ that is built from a (sub)set of $\mathcal{W}^j$. Then, $\rho' \geq \rho^{j'} \geq \rho^j = \rho$, where $\rho^{j'}$ is the bottleneck cost of $\mathcal{L}^{j'}$. Consequently, the multiway join plan $\mathcal{P}'$ incurs cost not lower than ρ. The proof is similar when services are removed from any other subplan. □

The second case, where the second phase of 2P-SO is not triggered, is when the plan built from the services in $\mathcal{W}^k$ does not have any service after the multi-way join service S^k_{join}, i.e., $\mathcal{C} = \emptyset$. The reason for not employing the second phase is that we do not expect any performance improvement, based on the observations described at the beginning of the section (first observation in Sect. 4.1). Theorem 2 formally states the above observation.

Theorem 2 *When after the first phase of the algorithm $\mathcal{C} = \emptyset$, then the second phase of 2P-SO does not lead to further performance improvements.*

Proof Suppose that the second phase of the proposed algorithm is triggered. Then one service, say S^i_r, is appended to $\mathcal{C}$, i.e., $\mathcal{C} = \{S^i_r\}$. Let ρ' be the maximum cost incurred by the multi-way join plan after the employment of the second phase of the algorithm.

If S_r^i is moved back to the end of the corresponding subplan, i.e., $\mathcal{L}^i$, then the maximum cost incurred by the multi-way join plan after that change is not higher than ρ' due to the following reasons. First, $T_{join,r}^{k,i} \geq T_{r,join}^{i,k}$, since the computation costs are negligible and the join selectivity is higher than 1. Second, the cost incurred by the join service S_{join}^k is not increased, since the latter service does not send data to another service and processes fewer data (more data is filtered out before, because more filtering services are placed before S_{join}^k). The above proves that the second phase cannot improve the performance of the initially found plan. □

The last case, in which no further improvement to the initial multi-way join plan can be achieved, is when (i) the bottleneck service S_l^j is in a linear subplan $\mathcal{L}^j$, $j \neq k$, (ii) its immediate successor is the multi-way join service and (iii) the cost that it incurs obeys the following condition

$$\prod_{i|S_i^j \in \mathcal{W}^j \wedge S_i^j \neq S_l^j} \sigma_i T_{l,join}^{j,k} \leq \prod_{i|S_i^j \in \mathcal{W}^j \wedge S_i^j \neq S_r^j} \sigma_i T_{r,join}^{j,k}, \tag{2}$$

where $S_r^j \neq S_l^j$. The left part of Eq. (2) corresponds to the cost that service S_l^j incurs when it is placed just before the multiway join service S_{join}^k in the linear subplan $\mathcal{L}^j$. The right part of Eq. (2) corresponds to the cost incurred by another service S_r^j in an alternative subplan $\mathcal{L}^{j'}$ of the same set of services $\mathcal{W}^j$, where S_r^j is now the service that is placed just before the multiway join service. Equation (2) states that in any linear subplan $\mathcal{L}^{j'}$ from the services in $\mathcal{W}^j$ the cost a service S_r^j incurs in order to process input data and send the results to the multiway join service is not lower than the cost incurred by service S_l^j in subplan $\mathcal{L}^j$ when S_l^j is placed just before the multiway join service.

Theorem 3 *Let $\mathcal{L}^j$, $j \neq k$ be the subplan that incurs the maximum cost in a multi-way join plan that is built during the first phase of the proposed algorithm. Suppose that the bottleneck service S_l^j of $\mathcal{L}^j$ is also the last service in $\mathcal{L}^j$ and that it satisfies Eq.* (2) (*see Fig.* 6a). *Then, the second phase of 2P-SO cannot improve the performance of the initially found multi-way join plan.*

Proof Without loss of generality, let $\mathcal{L}^{j'}$ be another plan that is formed by a different ordering of the services in $\mathcal{W}^j$, see Fig. 6b. Let S_r^j be the last service in $\mathcal{L}^{j'}$. Then the cost that is incurred by S_r^j is not lower than the cost incurred by S_l^j in $\mathcal{L}^j$ due to Eq. (2), and thus, the bottleneck cost of $\mathcal{L}^{j'}$ is not lower than the bottleneck cost of $\mathcal{L}^j$. If we remove any service from $\mathcal{L}^{j'}$ that is placed before S_r^j, say S_s^j, then the cost that is incurred by S_r^j in the new plan is higher now, since the services in $\mathcal{W}^j$ are selective (see Fig. 6c). This means that any other subplan that is produced after the employment of the second phase cannot incur a cost that is lower than the maximum cost incurred by the initially found subplan $\mathcal{L}^j$. Consequently, the bottleneck cost of the plan $\mathcal{L}^j$ cannot be improved after removing any service from it, and thus, the second phase does not lead to further performance improvements. □

$$\begin{tikzcd}
 & & & & & \text{bottleneck service} \arrow[d] & \\
X^j \arrow[r] & S_1^j \arrow[r] & S_2^j \arrow[r] & \ldots \arrow[r] & \ldots \arrow[r] & S_l^j \arrow[r] & S^k_{join}
\end{tikzcd}$$

(a)

$$\begin{tikzcd}
X^j \arrow[r] & S_i^j \arrow[r] & \ldots \arrow[r] & S_s^j \arrow[r] & \ldots \arrow[r] & S_r^j \arrow[r] & S^k_{join}
\end{tikzcd}$$

(b)

$$\begin{tikzcd}
X^j \arrow[r] & S_i^j \arrow[r] & \ldots \arrow[r] & \ldots \arrow[r] & S_r^j \arrow[r] & S^k_{join}
\end{tikzcd}$$

(c)

Fig. 6 **a** The subplan $\mathcal{L}^j$ that is initially built. The service S_l^j is the bottleneck one. **b** A subplan $\mathcal{L}^j$ formed by a different ordering of the services in $\mathcal{W}^j$, where S_r^j is placed at the end. **c** The subplan $\mathcal{L}^{j'}$ after removing a service S_s^j

5 Optimization for non-physically clustered services

The experimental results (see Sect. 6) show that 2P-SO can produce in the majority of cases an optimal multiway join plan when the services are logically clustered and Eq. (1) holds. However, in the general setting where the aforementioned assumptions are violated, Theorem 1, which proves when an optimal plan is found during the first phase of 2P-SO, is satisfied a significantly lower number of times. This limitation has driven us to generalizing 2P-SO. The generalization concerns two points: (i) the choice of services that will be removed from a linear subplans $\mathcal{L}^j$ and placed after the multiway join service and (ii) the number of plans that are visited before reaching to a solution.

In this section, we present a new theorem that (i) derives when it is possible to reduce the cost of the bottleneck subplan $\mathcal{L}^j$ after a multiway join plan $\mathcal{P}$ is created and (ii) provides a candidate set of services that should be removed and appended to $\mathcal{C}$. This theorem is independent of the assumptions that Eq. (1) holds and forms the basis of a more general query optimization algorithm, called *Advanced 2 Phase-Service Ordering* (A2P-SO), that is presented in Appendix B. A disadvantage of the proposed theorem is that the computations that must be performed to check its validity are more expensive than those of Theorems 2 and 3. Another disadvantage is that it is intrusive with regards to the linear ordering algorithm that was previously used as a black box and it is unclear whether any other linear ordering algorithm can be used instead.

As will be shown in Sect. 6, an advantage of A2P-SO is that it can build plans with lower response time than 2P-SO without incurring significantly larger runtime overhead.

The section starts with the rationale behind the new theorem, continues with a short presentation of the algorithm in [18] that is the basis of the new theorem and ends with the theorem's description. Table 2 summarizes the notation used throughout Sect. 6.4.

Table 2 Notation used in Sect. 5

Symbol	Description
$\mathcal{O}$	An optimal linear plan of services built by the algorithm in [18]
$\mathcal{V}$	A partial linear plan of services visited by the algorithm in [18] before convergence to $\mathcal{O}$
ϵ	The bottleneck cost of a partial plan $\mathcal{V}$
$\overline{\epsilon}$	The maximum cost that may be incurred by the services that are not currently included in $\mathcal{V}$
Φ^j	A subset of services, such that a subplan $\mathcal{L}^{j'}$ that is built from the services in $\mathcal{W}^j - \Phi^j$ has bottleneck cost lower than the bottleneck cost of $\mathcal{L}^j$
$\mathcal{V}^j$	A partial linear plan of services visited by the algorithm in [18] before convergence to $\mathcal{L}^j$

Let $\mathcal{L}^j$ be the bottleneck subplan of $\mathcal{P}$, which incurs cost ρ^j. For the moment, assume that $\mathcal{W}^j$ does not contain the multi-way join service S^k_{join}. A necessary step towards creating a new multiway join plan with lower bottleneck cost is to find a subset of services from $\mathcal{W}^j$, denoted by Φ^j, such that the subplan $\mathcal{L}^{j'}$ that comprises the services in $\mathcal{W}^j - \Phi^j$ has bottleneck cost lower than ρ^j. Note that the cost incurred by the subplan $\mathcal{L}^{j'}$ incorporates the cost spent by the last service in $\mathcal{L}^{j'}$ to process incoming tuples and to send the results to the join service. In cases where $\mathcal{L}^k$ is the bottleneck subplan, Φ^k must contain the multi-way join service.

5.1 Brief description of the linear ordering algorithm

The algorithm for building optimal linear pipelined plans follows the branch-and-bound optimization paradigm [18]. This algorithm is capable of efficiently exploring the solution space employing several criteria. With the help of that criteria, the algorithm can find the optimal solution without necessarily building the entire linear plan. Particularly, the algorithm is capable of detecting when a partial plan has the potential to be part of the optimal solution, and thus, the algorithm should continue its exploration by appending new services at the end of it. On the other hand, when the algorithm detects that the currently explored partial plan cannot lead to the optimal solution, it prunes it with a view to exploring additional plans.

Starting with an empty plan $\mathcal{V}$ and an empty optimal linear plan $\mathcal{O}$ with infinite bottleneck cost, in every iteration of the algorithm, the parameters ϵ and $\overline{\epsilon}$ are computed. ϵ corresponds to the bottleneck cost of $\mathcal{V}$, while $\overline{\epsilon}$ is the maximum possible cost that may be incurred by services not currently included in $\mathcal{V}$. If all selectivities are less than 1, $\overline{\epsilon}$ is given by:

$$\overline{\epsilon} = \max_{l,r} \left\{ \begin{array}{ll} \left(\prod_{j|S_j \in \mathcal{V}} \sigma_j\right) T_{l,r}, & S_l \notin \mathcal{V},\ S_r \notin \mathcal{V} \\ \left(\prod_{j|S_j \in \mathcal{V} \wedge j \neq l} \sigma_j\right) T_{l,r}, & S_l : \mathit{last\ service\ in}\ \mathcal{V},\ S_r \notin \mathcal{V} \end{array} \right\} \quad (3)$$

If there exists a service S_i with $\sigma_i > 1$ (which is S^k_{join} in our case), then $\overline{\epsilon}$ in Eq. (3) is multiplied by the product of all $\sigma_i > 1$ such that $S_i \notin \mathcal{V}$.

If the bottleneck cost ϵ of $\mathcal{V}$ is lower than $\overline{\epsilon}$, this means that the bottleneck cost of the plan beginning with $\mathcal{V}$ depends on the ordering of the services not yet included; so a new service is appended to $\mathcal{V}$. We select the service having the minimum aggregate cost with respect to the last service. Whenever the conditions $\overline{\epsilon} \leq \epsilon$ and $\epsilon < \rho$ are met, a new solution is found. The above condition implies that the ordering of the services that are not yet included in $\mathcal{V}$ does not affect its bottleneck cost. So, a candidate optimal solution $\mathcal{O}$ is found, with bottleneck cost $\rho = \epsilon$, that consists of the current partial plan $\mathcal{V}$ followed by the rest of the services in any order. Finally, if the bottleneck cost ϵ of the current plan $\mathcal{V}$ is higher than or equal to the bottleneck cost ρ of the best plan found so far $\mathcal{O}$, then $\mathcal{V}$ is pruned. In that case, $\mathcal{V}$ cannot yield an optimal solution, since its bottleneck cost is not lower than the bottleneck cost of $\mathcal{O}$. The algorithm can safely terminate when the less expensive pair of services that are not placed at the beginning of any plan already visited incurs cost at least as high as the bottleneck cost ρ of the currently best solution $\mathcal{O}$ [18].

5.2 On reasoning about the effectiveness of A2P-SO

The partial plans $\mathcal{V}^j$ that are produced by the algorithm in [18] when building the bottleneck subplan $\mathcal{L}^j$, can be exploited to check if there exists a subplan $\mathcal{L}^{j'}$ that incurs less cost than the cost of $\mathcal{L}^j$. It is assumed that $\mathcal{W}^j$ does not contain S^k_{join}, however, the following analysis is similar when $\mathcal{W}^j = \mathcal{W}^k$.

Let $\mathcal{V}^j$ be a partial linear plan that is produced utilizing the services in $(\mathcal{W}^j \cup S^k_{join})$, before the algorithm in [18] converges to $\mathcal{L}^j$ (see Sect. 4). Consider that $S^k_{join} \notin \mathcal{V}^j$. We are interested only in partial linear plans $\mathcal{V}^j$ that satisfy the following two conditions[10]: $\epsilon < \overline{\epsilon}$ and $\epsilon < \rho^j$. Based on such a partial plan, we want to create a subplan $\mathcal{L}^{j'}$, which consists of the current partial plan $\mathcal{V}^j$ placed at its beginning followed by an *appropriate subset* of the rest services, such that $\rho^{j'} < \rho^j$ (recall that $\mathcal{V}^j$ satisfies the conditions $\epsilon < \overline{\epsilon}$ and $\epsilon < \rho^j$). In other words, we search for those services that if they are removed from $\mathcal{W}^j$, a new subplan $\mathcal{L}^{j'}$ is formed with bottleneck cost $\rho^{j'} < \rho^j$.

If, for a partial linear plan $\mathcal{V}^j$ that satisfies the aforementioned conditions, the cost spent by the last service in $\mathcal{V}^j$ to process incoming data and to send the result tuples to the multi-way join service S^k_{join} is lower than the bottleneck cost of $\mathcal{L}^j$ (ρ^j), i.e.,

$$cost(S^j_l) = R^j_l(\mathcal{V}^j)T^{j,k}_{l,join} < \rho^j, \quad S_l : \mathit{last\ service\ in}\ \mathcal{V}^j \tag{4}$$

then there exists a subset of services Φ^j, and consequently a subplan $\mathcal{L}^{j'}$ with bottleneck cost less than the bottleneck cost of $\mathcal{L}^j$. For the proof, see Lemma 1 in the Appendix.

Note that if $\epsilon < \overline{\epsilon}$ and $\epsilon < \rho^j$ are satisfied for a partial plan $\mathcal{V}^j$ and $S^k_{join} \in \mathcal{V}^j$, then a linear subplan $\mathcal{L}^{j'}$ with bottleneck cost less than the bottleneck cost of $\mathcal{L}^j$ can be also created (see Lemma 4 in Appendix 9).

Theorem 4 summarizes under which cases, given a multiway join plan $\mathcal{P}$, we can produce a new multiway join plan $\mathcal{P}'$ for which the cost incurred by its subplans $\mathcal{L}^{j'}$ is lower than the cost of the subplans $\mathcal{L}^j$ of plan $\mathcal{P}$. Its proof is in the Appendix.

[10] Details are provided in the Appendix.

Theorem 4 *Let $\mathcal{V}^j$ be one of the partial plans that are visited by the algorithm in* [18] *when building the bottleneck subplan, which is denoted by $\mathcal{L}^j$. Also assume that $\mathcal{V}^j$ satisfies $\epsilon < \overline{\epsilon}$ and $\epsilon < \rho^j$. If either of the following conditions hold*

1. *$S^k_{join} \notin \mathcal{V}^j$ and Eq. (4) is satisfied, or*
2. *$S^k_{join} \in \mathcal{V}^j$,*

then there exists a subset of services $\mathcal{W}^j - \Phi^j$ from which a linear subplan $\mathcal{L}^{j'}$ can be formed with bottleneck cost lower than the bottleneck cost of the initially found subplan $\mathcal{L}^j$. Otherwise, any subplan $\mathcal{L}^{j'}$ that will be formed by any subset of services in $\mathcal{W}^j$, will incur cost at least as high as the bottleneck cost of $\mathcal{L}^j$.

In order to use Theorem 4, we must perform a few modifications to the algorithm in [18], such that when a partial linear plan $\mathcal{V}^j$ is visited that satisfies $\epsilon < \overline{\epsilon}$ and $\epsilon < \rho^j$, the Eq. (4) must be checked.

It must be emphasized that Theorem 4 chooses the services that should be removed from $\mathcal{L}^j$ and does not guarantee that after appending the services in Φ^j to $\mathcal{C}$, the new maximum cost incurred in $\mathcal{C}$ does not increases.

6 Evaluation

The evaluation of the proposed algorithms are based on extensive simulations of multiple-input join queries. Overall, five aspects are investigated. First, we assess the benefits incurred by the application of the second phase of 2P-SO as compared to the case when only the first phase is performed. In particular, we study how often the second phase of 2P-SO is triggered and the percentage of the performance improvements after the application of the second phase. To this end, two different sets of simulation instances are generated that differ with respect to the selectivity of the filtering services. The second aspect that is investigated is the impact on the performance of the multi-way join plan of the single-input linear ordering algorithm proposed in [18] against other simpler approaches to single-input linear ordering, namely the bottleneck traveling salesman (BTSP) algorithm [21]. Third, the efficiency of 2P-SO in terms of the maximum number of iterations performed by [18], in order to build each of the $\mathcal{L}^j$ subplans is studied. Fourth, we compare 2P-SO and A2P-SO in terms of the response time of the resulting plans and, finally, we observe the running times of the proposed algorithms.

6.1 Experimental setting

The simulation environment is produced as follows. We consider queries over 2, 5 and 10 data resources, where each resource is associated with a different set of services, as described in the previous sections. We are mostly interested in medium and large scale pipelined queries; as such, the number N of services $\mathcal{W}^j$ is varied between 5 and 100 services $\{5, 10, 20, \ldots, 100\}$. It is assumed that all data resources produce equal number of tuples. The selectivity of the join service in both instance sets lies between 1 and 10, while the selectivity of the filtering services lie in the interval [0.1 1].

The per tuple transferring costs between the services satisfy the triangle inequality, while the per tuple processing cost is much lower than the communication cost. In particular, the processing cost c_i^j for each S_i^j follows a normal distribution with mean value $\overline{c} = 1$ and standard deviation 0.1. The per tuple data transmission costs $t_{i,r}^{j,j}$ between services of the same set follow a normal distribution, too, with mean value $\overline{t}_{intra}$ 10 or 100 or 1,000 times $\overline{c}$ and standard deviation equal to $0.2\overline{t}_{intra}$. In this way, even within a service cluster, the communication cost dominates. The per tuple transferring cost between services belonging to different sets is much higher than the cost between services of the same set. The data transmission costs $t_{i,r}^{j,l}$ follow a normal distribution with mean value $\overline{t}_{inter}$ either 10 or 100 times $\overline{t}_{intra}$, while the standard deviation is set to $0.1\overline{t}_{inter}$. We ensure that Eq. (1) is always satisfied. Later, we are going to relax this assumption.

The above inter- and intra-cluster standard deviation values are appropriately selected in order to reflect the statistics that are collected from PlanetLab in [18]. According to the observed inter-service communication times, the simulation instances that are produced using $\overline{t}_{inter} = 10\overline{t}_{intra}$ better reflect computational clusters that are located in neighboring countries and communicate via high-speed links, e.g., one cluster is located in Spain the other is in Portugal, while the instances having $\overline{t}_{inter} = 100\overline{t}_{intra}$ better simulate computational clusters that are located in different continents. The $\overline{t}_{intra} = 10\overline{c}$ values, in turn, are used in order to better simulate applications that perform some kind of processing on the incoming data, e.g., applications that extract features from input images, while the $\overline{t}_{intra} = 100\overline{c}$ and $\overline{t}_{intra} = 1,000\overline{c}$ values reflect cases where no intensive processing is done on the incoming data, e.g., a filtering service that simply checks whether the value of a numerical attribute of an incoming tuple is above a threshold or not.

For each such setting, 10 different random instances are produced. Overall, for each technique or algorithm flavor, we produce $3 \times 11 \times 3 \times 2 \times 10 = 1{,}980$ different plans corresponding to random queries, ten for each combination of number of data resources, services per set, intra-set average communication cost and inter-set average communication cost.

6.2 Performance benefits of the second phase of 2P-SO

In the first experiment, it is studied how often the second phase of 2P-SO is triggered, as well as the obtained performance benefits. There, we have observed that the second phase is triggered only for the 1.25 % of the random instances. In particular, for the 82.7 % of the instances, the bottleneck subplan $\mathcal{L}^j$ incurs the minimum cost among the subplans that can be built from any subset of $\mathcal{W}^j$, which means that the produced multi-way join plan is guaranteed to be optimal (see Theorem 1). This phenomenon is attributed to the facts that (i) the values $T_{i,r}^{j,j}$ do not have significant deviations (i.e., deviations that exceed an order of magnitude) and (ii) the selectivity of the filtering services is uniformly distributed between 0.1 and 1. For the rest 12.85 % of the instances, no service is added to $\mathcal{C}$ during the first step of 2P-SO, and thus Theorem 2 is satisfied. This phenomenon is attributed to the fact that $T_{join,i}^{k,j} \geq T_{i,r}^{j,j}$

and $T^{k,k}_{join,i} \geq T^{k,k}_{i,join}$. As a result, the algorithm in [18] selects to place at the end of a subplan the multi-way join service, in order to reduce as much as possible the aggregate cost that it incurs.

In 3.2 % of the cases, the bottleneck service S^j_{bottl} is the immediate predecessor of the multi-way join service S^k_{join} and $j \neq k$. For those instances, we have observed that (i) the inter-cluster communication cost is, approximately, two orders of magnitude higher than the intra-cluster communication cost (the instances have $t_{inter} = 100t_{intra}$) and (ii) the number of services per set is 5 or 10. Since the number of filtering services is relatively small and the intra-cluster $T^{j,j}_{i,r}$ values are two orders of magnitude lower than the inter-cluster $T^{j,l}_{i,r}$ values, the cost that is incurred by the service that precedes the join is high, relatively to the cost incurred by the rest services in $\mathcal{L}^j$. Thus, the algorithm that is utilized for building the subplans $\mathcal{L}^j$ chooses to place just before the multi-way join service S^k_{join} the service that incurs the minimum aggregate cost with S^k_{join}. The second phase of 2P-SO is not triggered for the above instances, since, according to Theorem 3 (Sect. 4), it cannot improve the initial bottleneck cost.

For the rest 1.25 % of the instances, where the second phase of 2P-SO is triggered, the bottleneck cost of the initially found plan is improved for the 43 % of these instances, i.e., the 0.5375 % of the 1,980 random instances. In the latter instances, where performance improvements are observed, the bottleneck subplan was formed by the service set that contained the join service. The mean performance improvement is 25 %. Note that for the rest 0.7125 % of the 1,980 random instances, where the second phase did not improve the initially found bottleneck cost, none of the partial linear plan $\mathcal{V}^j$ with $\epsilon < \overline{\epsilon}$ and $\epsilon < \rho^j$ satisfied either of the conditions (1) and (2) of Theorem 4 (see Sect. 5). Figure 7 summarizes the results of the first experiment.

We have repeated the same experiment with a new set of 1,980 instances that are produced as described in Sect. 6.1. However, in the new instance set, the selectivity of the filtering services lies in [0.5 1]. The experimental results do not show significant differences. In 87 % of the cases, 2P-SO found the optimal multi-way join plan, while the second phase is triggered for the 2.7 % of the random instances. The performance improves for 26 % of the latter instances and the mean performance improvement is 28 %. Similar to the instances of the first set (where the services are more selective), for the cases, where the second phase successfully improved the initial bottleneck cost, the bottleneck subplan was formed by the services in $\mathcal{W}^k$.

6.3 Impact of the choice of the linear ordering algorithm

In the experiments presented so far, the single-input linear ordering of all $\mathcal{L}^j$ subplans is done according to the algorithm in [18]. Intuitively, it is expected that the algorithm employed for single-input data resource queries, which is the main building block of the complete algorithm, has a significant impact on the query response time. The third part of the experiments investigates the behavior of 2P-SO when the single-input ordering is performed according to the BTSP [21] algorithm. Note that this solution was initially proposed for slightly different problems. The reason we chose that for comparison is that we are not aware of any other solution for exactly the same

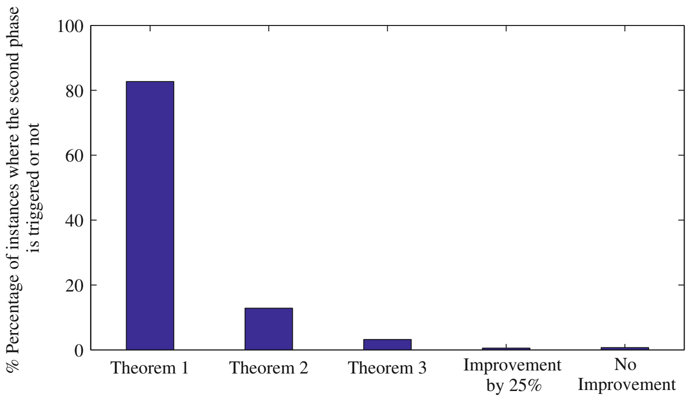


Fig. 7 The results of the first experiment

problem apart from [9]. In [18], the performance of the algorithm in [18] is compared with the Greedy algorithm of [9] in a real-word, wide-area setting the PlanetLab [22]. There, the experimental results showed that the algorithm in [18] obtained significant performance improvements with respect to [9].

The solution in [21] deals with the BTSP problem. In BTSP, we search for a path visiting all vertices in a complete undirected graph, such that the largest weight of an edge in the path is as small as possible and the path visits each node exactly once. This problem definition is very similar to the problem of minimizing the response time of queries over a single data resource for the case where all selectivities are set to 1. Although many algorithms have been proposed for BTSP, in the conducted experiments, we utilize the algorithm in [21], which reduces BTSP to ordinary TSP problem. That algorithm is computationally efficient and provides a 2-approximation to the optimal solution when the edge weights of the graph satisfy the triangle inequality. In order to build a linear ordering $\mathcal{L}^j$ from the set $\mathcal{W}^j$ utilizing [21], a graph having as vertices the services in $\mathcal{W}^j$ with zero processing cost and selectivity equal to one is constructed. The edge weights are defined by the communication cost of the corresponding services. Since the BTSP algorithm needs a service (vertex) to start from, in order to find a linear plan using BTSP, we have built multiple linear plans using each time a different, randomly selected starting service. Then, among the produced plans we selected the one with the minimum response time.

In order to build a multi-way join plan utilizing the BTSP algorithm, the single-resource subplans $\mathcal{L}^j$ are built and then their output is connected with the multi-way join service of $\mathcal{L}^k$ (assuming that W^k is the set including the join service). The second phase of the algorithm proposed in Sect. 4, i.e., placing the bottleneck service of a subplan after the join service, is also applied, in order to further improve the performance of the resulting plans. The efficiency of this algorithm is measured in terms of the ratio of the response time of the multi-way plans it produces and the response time of the multi-way plans built by 2P-SO.

Figure 8a, b shows the ratio between the response time of the multi-way join plans that are built using the BTSP algorithm and the response time of the multi-way join plans built by 2P-SO, when the join service has two inputs and the size of $\mathcal{W}^j$ ranges

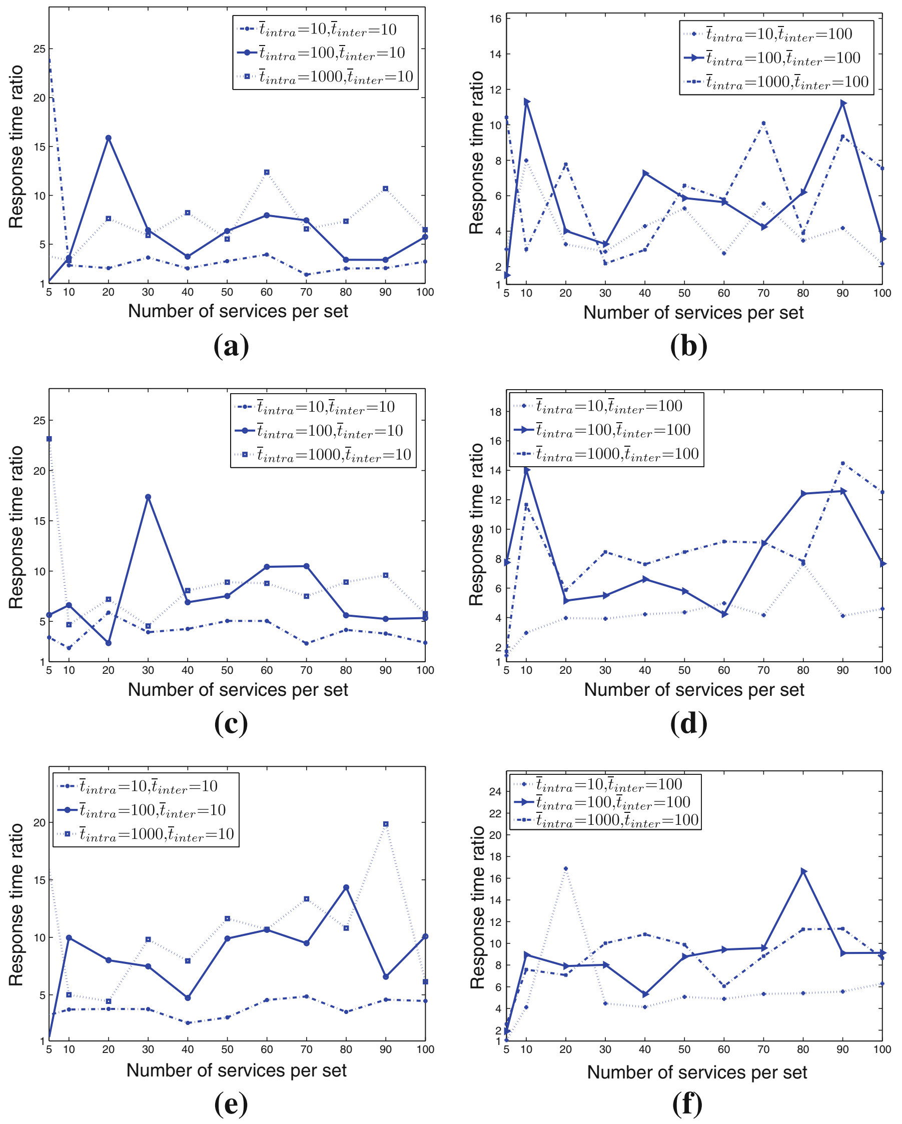


Fig. 8 The ratio between 2P-SO and the BTSP algorithm for different sizes of service clusters for queries over **a**, **b** 2; **c**, **d** 5; and **e**, **f** 10 data resources

in {5, 10, 20, . . . , 100}. Each plot in Fig. 8a, b corresponds to different mean intra-set and inter-set communication values. Recall that 10 different random instances were created with the same settings (see Sect. 6.1). Both the BTSP and 2P-SO are executed for all these random instances and Fig. 8a, b shows the median of the response time ratios calculated over the multi-way join plans produced from the 10 random instances that are created using the same settings. The fluctuations in the figures are attributed to the fact that the actual inter- and intra-cluster communication cost values in every

Table 3 Performance degradation while building plans with the BTSP algorithm without applying the second phase

Number of input resources	Mean performance degradation (%)
$L = 2$	50
$L = 5$	59
$L = 10$	51

simulation instance are randomly produced each time through a normal distribution generator. The following observations can be drawn:

- Employing the algorithm in [18] always outperforms the other solution (i.e., the ratio is always above 1).
- There are cases where the performance improves by an order of magnitude (e.g., 20 times) when the algorithm in [18] is employed.
- On average, the performance deviations tend to increase with (i) the increase of the per tuple mean transferring cost and (ii) the number of services per set.

The same observations can be drawn for 5-input (Fig. 8c, d) and 10-input (Fig. 8e, f) queries. Again, the observed fluctuations are a byproduct of the random generation of the actual inter- and intra-cluster communication cost values in every simulation instance. From Fig. 8a–f, we can observe that although the number of inputs, services or the per tuple transferring costs do not have a significant impact on the magnitude by which 2P-SO outperforms the BTSP one, 2P-SO has always better performance, i.e., the ratio between the response time of the plans built by the BTSP algorithm and 2P-SO are constantly higher than one.

Recall that Fig. 8a–f shows the performance of the join plans that are built using the BTSP algorithm, after employing a two-phase approach as presented in Sect. 4. Table 3 summarizes the mean performance degradation for different number of input data resources L, after building multi-way join plans using the BTSP algorithm, but without employing the refinement phase of Sect. 4. We can see that, in contrast to 2P-SO, the second phase is obligatory, in order to obtain feasible join plans using the BTSP algorithm.

The comparison of the two methods for building multiway join plans as presented in Fig. 8a–f is in relative time units. We have selected such a representation in order to show the importance of 2P-SO as the per tuple processing and transferring costs increase. For example, if the per tuple processing and transferring costs of the filters are in the order of seconds, then a plan that will be created utilizing the BTSP algorithm will have on average tens of dozens higher response time than a plan built by 2P-SO. The low running time of 2P-SO in order to build multiway join plans (see Sect. 6.5) further supports its significance.

Figure 9a, b shows the maximum number of iterations that are performed to build the $\mathcal{L}^j$ subplans of the Fig. 8a–f.

Counter-intuitively, there are some cases where the number of iterations is higher although the number of services is lower. This must attributed to the way the algorithm in [18] operates. In the following, we will provide a brief explanation of this phenomenon. When there exist a few filtering services before the join service, the output selectivity of the join service, i.e., the product of the selectivities of the services

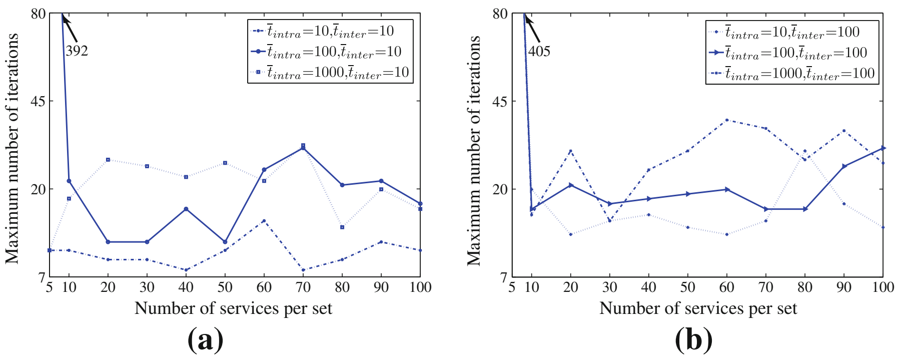


Fig. 9 The maximum number of iterations required by our algorithm to build $\mathcal{L}^j$ subplans

of a linear subplan $\mathcal{L}^j$ and the selectivity of the join service with respect to the input X^j, is relatively high. Thus, a parameter that controls the stopping condition of the algorithm in [18] decreases slowly, forcing the branch and bound algorithm to perform many iterations until convergence (see Sect. 5). In general, we observe that the number of the performed iterations does not change significantly with the number of the services. Also, the differences in the inter-service communication times do not have a significant impact, either.

6.4 Evaluation of 2P-SO and A2P-SO when services are only logically clustered

Finally, we have studied the performance of 2P-SO and A2P-SO with one more set of random instances, for which the processing cost of data is higher than the intra-cluster communication cost. The instances were produced following an approach similar to the one employed in Sect. 6.1. However, for the new set of random instances, we have set $\bar{c} = 5$, $\bar{t}_{intra} = 1$, while $\bar{t}_{inter}$ is 5 or 20 times $\bar{t}_{intra}$. The standard deviations of the distributions were the same with Sect. 6.1. Note that using the above settings, Eq. (1) is not always satisfied, i.e., this case is similar to scenarios where services are only logically clustered. The parameters of A2P-SO as shown in Fig. 11 in Appendix B are $N = 10$, $M = 2$ and $iter = 1{,}000$.

When employing 2P-SO, for the 12.3 % of the random instances, Theorem 3 is satisfied. Note that Theorem 3 is valid independently of the assumption that the communication cost dominates the processing one and only depends on the assumption that all services, apart from the multi-way join service, are selective. 2P-SO managed to improve the bottleneck cost for the 69.2 % of the random instances during its second phase, where the mean performance improvement is 45.7 %. For the same set of random instances, A2P-SO built plans with 37.5 % lower response time compared to the plans of 2P-SO on the average. For the rest 18.5 % of the instances, the second phase of the proposed algorithms did not lead to performance improvements either because the algorithm reached to the optimal solution during the first phase or due to reasons explained through Theorem 4; for those instances, none of the partial plans $\mathcal{V}^j$ with $\epsilon < \bar{\epsilon}$ and $\epsilon < \rho^j$ satisfied either of the conditions (1) and (2) of Theorem 4.

6.5 Running time of 2P-SO and A2P-SO

We have also investigated the running time of 2P-SO and A2P-SO. All experiments were performed on a i5 Intel Linux machine. The mean running time of 2P-SO and A2P-SO per simulation instance is well below one second (approximately 30 ms for 2P-SO and 70 msec for A2P-SO).

6.6 Discussion

To summarize, the results of the experiments show that when the services are both logically and physically clustered, 2P-SO can find the optimal multiway join plan during the first phase in most of the cases (82.7 % of the generated random instances). For the 16.75 % of the generated random instances the second phase did not lead to further performance improvements due to Theorems 2–4, while, for the rest 0.53 % of the generated random instances the second phase improved the performance of the multiway join plans by 25 % approximately. Another observation is that 2P-SO builds multiway join plans having better response time (up to an order of magnitude) than the multi-way join plans built using the BTSP algorithms.

When the services are only logically clustered, A2P-SO is proven to be a better choice than 2P-SO as it builds plans with lower response time (by 37.5 % approximately compared to the plans built by 2P-SO). In contrast to cases where the services are both logically and physically clustered, the second phase of 2P-SO can lead to performance improvements of 45.7 % on the average in the 69.2 % of the random instances.

From the above discussion we can conclude the following: (i) the second phase of 2P-SO can lead to more efficient multiway join plans and (ii) 2P-SO is a good candidate even in cases where Eq. (1) is violated as it builds efficient multiway joins with approximately 2.5 lower running time than A2P-SO. However, when the response time of a plan is of primary importance, A2P-SO is more suitable for optimization under a more general setting, as it outputs plans with 37.5 % on the average lower response time than 2P-SO.

7 Related work

The problem of query optimization with a view to minimizing the response time while employing pipelined parallelism in wide area environments has been largely overlooked; most of the work proposed in the literature considers that a plan's response time equals to the total cost incurred by all operators in that plan [23]. In contrast, when pipelined parallelism is employed the response time depends on the bottleneck operator, i.e., the operator that incurs the maximum cost among the rest of the operators in the plan to process input data. Pipelined operator ordering bears several similarities with optimizing pipelined calls to services, and 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 operators are independent [24]. In a wide-area environment, Srivastava et al. [9] proposed an algorithm for optimizing single-input queries over services. Moreover, this work is based on the assumption that the transferring cost of data is negligible. Babu et al. [16] developed a solution for the ordering of filtering operators that are tailored to online, dynamic scenarios. However, the proposed approximate algorithm applies to the problem of minimizing the total work and, apart from that, the transferring cost of tuples is considered to be negligible.

A common characteristic of the proposals mentioned so far is that they build a single plan. For completeness, we mention techniques that build multiple plans in order to maximize the data flow, which is equivalent to minimizing the bottleneck cost. In [25], such an algorithm is proposed in order to maximize the flow of tuples processed by a number of filtering and unconstrained operators. The output of the algorithm is a set of linear plans, each one of them having assigned to a weight. When a new tuple enters the system, it is probabilistically routed to one of these linear plans according to the assigned weights. It must be noted that the flow maximization algorithm considers only the processing cost of the operators and the potentially heterogeneous communication costs are again disregarded. This work is extended in [15] to support proliferative operators and precedence constraints. However, Ref. [15] is characterized by the limitation of not considering communication costs, too. Existing solutions for multi-query optimization (e.g., [26]) neglect the communication cost between the operators, as well.

Our work relates to the broader areas of distributed query optimization, as well. Distributed query optimization algorithms typically refer to problems of high computational complexity [8]. They differ from their centralized counterparts in that the communication cost must be also considered. The adoption of parallelism during query execution makes the problem of query optimization even harder [27]. This has leaded to the adoption of more sophisticated dynamic programming techniques (e.g., [28]) or heuristics. A recent work in the area of distributed query optimization appears in [29]. That work deals with the problem of optimizing multiple, overlapping (by means of the input data), non-parallel queries. The input data are distributed to different host machines, while the cost needed to transfer data between any pair of hosts is not negligible, as in our problem. Nevertheless, the optimization goal is different; the algorithm in [29] aims to minimize the sum of the total cost to transfer data across overlapping queries, whereas we focus on minimizing the response time of a single pipelined parallel query over multiple input data resources.

8 Conclusions

In this work, we deal with the optimization of pipelined multi-join queries over clustered filtering services in communication-bounded wide-area environments. Optimization of such queries is of high significance for large-scale data management involving remote access to services (e.g., web services that filter data and perform look-up operations). It is particularly relevant to scenarios where the order in which service calls are performed is flexible, and different orderings yield different execution times. Our optimization objective is the minimization of response time, which, due to pipelining, is determined by the slowest service in the plan.

The main novelty of our work is that we propose two efficient and effective optimization algorithms for queries that require multiple data inputs to be processed by distinct clusters of filtering services and also to be joined together. The execution environment that we assume allows services to communicate directly with each other, i.e., inter-service communication does not occur via a central component. We also provide a theoretical analysis of their performance. To the best of our knowledge, our work is the first proposal for this problem.

There are several directions for future work. First, in our algorithms, the service and network characteristics remain stable. However, especially for long running queries, those characteristics may be subject to changes during execution. Another avenue for future work is to couple service ordering and allocation decisions; in this case, the services are not allocated to predefined hosts but their location is chosen at runtime. Finally, in this work, the focus has been mostly on pipelined parallelism. Several workflow systems have developed efficient mechanisms for complementary forms of parallelism, such as partitioned parallelism. The development of efficient query optimizers for service-based queries taking into account all parallelism types is an interesting and challenging open issue.

9 Appendix A: Proof of Theorem 4

In Appendix A, we present the lemmas that are used to prove Theorem 4. For the following lemmas let $\mathcal{V}^j$ be a partial linear plan, which is visited by the algorithm in [18] when building the bottleneck subplan $\mathcal{L}^j$. Also assume that $S^k_{join} \notin \mathcal{V}^j$, unless it is stated otherwise (as in Lemma 4). It is also assumed that $\mathcal{W}^j$ does not contain S^k_{join}. The proofs are similar when $\mathcal{W}^j = \mathcal{W}^k$.

Lemma 1 *If a partial linear plan $\mathcal{V}^j$ satisfies Eq. (4) and conditions $\epsilon < \overline{\epsilon}$ and $\epsilon < \rho^j$, then there exists a subset of services Φ^j, and consequently, a subplan $\mathcal{L}^{j'}$ with bottleneck cost less than the bottleneck cost of $\mathcal{L}^j$. $\mathcal{L}^{j'}$ is created by appending to $\mathcal{V}^j$ the rest of the services in $\mathcal{W}^j - \Phi^j$ in any order*[11].

Proof The proof starts with an iterative procedure for finding a set of services Φ^j. One possible set of services Φ^j can be found as follows: for every pair of services (S^j_i, S^j_r) or (S^j_i, S^k_{join}), where S^j_i is either the last service in $\mathcal{V}^j$ or $S^j_i \notin \mathcal{V}^j$ and $S^j_r \notin \mathcal{V}^j$, we estimate the *maximum* cost that would be incurred by that pair if it was appended to $\mathcal{V}^j$ (see [18]). More formally, that cost for a service pair (S^j_i, S^j_r) is

- $T^{j,j}_{i,r}$ multiplied by the selectivity of services in $\mathcal{V}^j$ that precede S^j_i up to S^j_i, if S^j_i is the last service in $\mathcal{V}^j$, or
- $T^{j,j}_{i,r}$ multiplied by the selectivity of all services in $\mathcal{V}^j$ and the selectivity of the join service with respect to X^j, if S^j_i is not the last service in $\mathcal{V}^j$, or

[11] In cases where $S^k_{join} \in \mathcal{W}^j$, then S^k_{join} is also added to Φ^j.

- $T^{j,k}_{i,join}$ multiplied by the selectivity of all services in $\mathcal{V}^j$, if S^j_i is not the last service in $\mathcal{V}^j$ and S^j_i sends data to S^k_{join}.

If that cost is higher than or equal to ρ^j we have the following cases:

- If none of the services S^j_i, S^j_r belongs to $\mathcal{V}^j$, then both services are added to Φ^j.
- If the first service in the pair (S^j_i, S^j_r) belongs to $\mathcal{V}^j$, then only the second service of the pair, i.e., S^j_r, is added to Φ^j.
- If the first service of the pair (S^j_i, S^k_{join}) does not belong to $\mathcal{V}^j$, then S^j_i is added to Φ^j. Note that in this case S^j_i cannot be the last service in $\mathcal{V}^j$, due to Eq. (4).

After finishing the above procedure for every pair of services (S^j_i, S^j_r) or (S^j_i, S^k_{join}) (where S^j_i is either the last service in $\mathcal{V}^j$ or $S^j_i \notin \mathcal{V}^j$ and $S^j_r \notin \mathcal{V}^j$), the services that are not included in Φ^j will incur cost less than ρ^j, independently of the order they are appended to $\mathcal{V}^j$. Thus, $\mathcal{L}^{j'}$ that is formed by appending to $\mathcal{V}^j$ the rest of the services in $\mathcal{W}^j - \Phi^j$ will incur cost less than ρ^j.

In the extreme case, where the maximum cost incurred by all service pairs apart from (S^j_l, S^k_{join}), where S^j_l is the last service in $\mathcal{V}^j$, is higher than or equal to ρ^j, then all services in $\mathcal{W}^j$ that are not included in $\mathcal{V}^j$ will be added to Φ^j and the bottleneck cost of $\mathcal{L}^{j'} = \mathcal{V}^j$ is ϵ. □

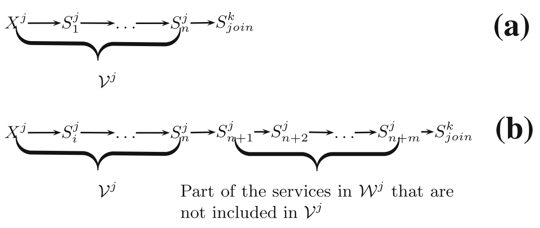
The following lemma explains why only the partial linear plans $\mathcal{V}^j$ that satisfy $\epsilon < \overline{\epsilon}$ and $\epsilon < \rho^j$, must be considered, in order to check the existence of a subplan $\mathcal{L}^{j'}$ that incurs cost lower than $\mathcal{L}^j$.

Lemma 2 *Only the partial linear plans $\mathcal{V}^j$ that satisfy $\epsilon < \overline{\epsilon}$ and $\epsilon < \rho^j$ must be considered, in order to check the existence of a subplan $\mathcal{L}^{j'}$ that incurs cost lower than the cost incurred by $\mathcal{L}^j$.*

Proof The reason for checking only the partial plans that satisfy $\epsilon < \overline{\epsilon}$ and $\epsilon < \rho^j$ is given below. If $\epsilon < \rho^j$ is violated, i.e., $\epsilon \geq \rho^j$, then any subplan $\mathcal{L}^{j'}$ that starts with $\mathcal{V}^j$ has bottleneck cost not lower than ρ^j due to the non-decreasing property of the bottleneck cost function. On the other hand, if $\epsilon < \overline{\epsilon}$ is violated, i.e., $\epsilon \geq \overline{\epsilon}$, then the algorithm in [18] has already reached a candidate solution, which starts with $\mathcal{V}^j$ (and may be followed by other services). However, since [18] finds the optimal solution, that candidate solution plan has bottleneck cost not lower than ρ^j.

The reason for checking only the partial plans that are visited by the algorithm until convergence and not any other partial plan not visited by the algorithm is the following. The algorithm in [18] terminates as soon as the less expensive pair of services that are not placed at the beginning of any partial plan already visited incurs cost $\geq \rho$ (see Sect. 5). Consequently, any subplan $L^{j'}$ starting with S^j_x, where (S^j_x, S^j_y) is any pair of services, which are not placed at the beginning of any partial plan visited by [18] before convergence to $\mathcal{L}^j$, incurs cost $\geq \rho^j$. □

Finally, we have to prove that if Eq. (4) is never satisfied for any of the partial plans $\mathcal{V}^j$ (that are created by [18] when building $\mathcal{L}^j$) satisfying $\epsilon < \overline{\epsilon}$ and $\epsilon < \rho$, then there is not any subset of services Φ^j, and consequently, a subplan $\mathcal{L}^{j'}$ with bottleneck cost less than the bottleneck cost of $\mathcal{L}^j$.



(a)
(b)
$\mathcal{V}^j$
Part of the services in $\mathcal{W}^j$ that are not included in $\mathcal{V}^j$

Fig. 10 **a** A partial linear plan $\mathcal{V}^j$ that satisfies $\epsilon < \overline{\epsilon}$ and $\epsilon < \rho^j$, but not Eq. (4). The *arrow* between the last service in $\mathcal{V}^j$ and the multi-way join service S^k_{join} denotes the cost that is incurred by the former service to process data and to send the results to S^k_{join}. **b** A partial plan $\mathcal{V}^{j'}$ that is formed by appending to $\mathcal{V}^j$ a (sub)set of services in $\mathcal{W}^j$ (that are not included in $\mathcal{V}^j$). The new partial plan $\mathcal{V}^{j'}$ satisfies both conditions $\epsilon < \overline{\epsilon}$ and $\epsilon < \rho^j$ and Eq. (4)

Lemma 3 *If, for none of the partial plans $\mathcal{V}^j$ that satisfy $\epsilon < \overline{\epsilon}$ and $\epsilon < \rho^j$, Eq. (4) is satisfied, then, there is not any subset of services Φ^j, and consequently, a subplan $\mathcal{L}^{j'}$ with bottleneck cost less than the bottleneck cost of $\mathcal{L}^j$.*

Proof Let $\mathcal{V}^j$ be a partial linear plan that satisfies $\epsilon < \overline{\epsilon}$ and $\epsilon < \rho^j$, but Eq. (4) is violated (Fig. 10a). Suppose that there exists a (sub)set of services in $\mathcal{W}^j$ that are not included in $\mathcal{V}^j$, and if they are appended to $\mathcal{V}^j$, a new partial plan $\mathcal{V}^{j'}$ is created (i) having cost $< \rho^j$ and also (ii) satisfying Eq. (4). The lemma is not correct if the above partial plan $\mathcal{V}^{j'}$ (that incurs cost $< \rho^j$) is not visited by the algorithm in [18] (Fig. 10b). However, this can never happen as explained in [18] because it violates the optimality of the algorithm. □

Until now, it was assumed that $S^k_{join} \notin \mathcal{V}^j$. When $\epsilon < \overline{\epsilon}$ and $\epsilon < \rho^j$ are satisfied and $S^k_{join} \in \mathcal{V}^j$, then there exists a set of services Φ^j such that if they are subtracted from $\mathcal{W}^j$, a linear subplan $\mathcal{L}^{j'}$ with bottleneck cost less than the bottleneck cost of $\mathcal{L}^j$ can be formed.

Lemma 4 *If a partial plan $\mathcal{V}^j$ that satisfies $\epsilon < \overline{\epsilon}$ and $\epsilon < \rho^j$ includes the join service S^k_{join}, then there exists a set of services Φ^j such that if they are subtracted from $\mathcal{W}^j$, a linear subplan $\mathcal{L}^{j'}$ can be created with bottleneck cost less than the bottleneck cost of $\mathcal{L}^j$.*

Proof Since $\mathcal{V}^j$ incurs cost $\epsilon < \rho^j$ and $S^k_{join} \in \mathcal{V}^j$, then a subplan $\mathcal{L}^{j'}$ can be formed by the ordering of the services in $\mathcal{V}^j$ that precede the join service S^k_{join}. □

Using the above lemmas, we can prove Theorem 4. Particulary, Lemmas 1 and 4 prove the first part of Theorem 4, i.e., if a partial linear plan $\mathcal{V}^j$ with $\epsilon < \overline{\epsilon}$ and $\epsilon < \rho^j$, also satisfies either of the conditions (1) and (2) of Theorem 4, then there exists a subplan $\mathcal{L}^{j'}$ with bottleneck cost lower than the bottleneck cost of the initially found subplan $\mathcal{L}^j$. The other two lemmas prove the second part of Theorem 4, i.e., if Eq. (4) is never satisfied for any of the partial plans $\mathcal{V}^j$ with $\epsilon < \overline{\epsilon}$ and $\epsilon < \rho^j$, then there does not exist a subplan $\mathcal{L}^{j'}$ with bottleneck cost less than the bottleneck cost of $\mathcal{L}^j$. Recall that Lemma 2 proves that it suffices to search for subplans with bottleneck cost lower than the bottleneck cost of $\mathcal{L}^j$ only to the partial plans $\mathcal{V}^j$ with $\epsilon < \overline{\epsilon}$ and $\epsilon < \rho^j$.

10 Appendix B: A two-phase algorithm accounting for not physically clustered services

Appendix B aims to present a variant of 2P-SO, called A2P-SO, that accounts for multi-way join plan creation in environments where Eq. (1) is violated. A2P-SO consists of two phases, as well, however the second phase is performed differently; there, the

```
Inputs
  $\mathcal{W}^j, j = 1, \ldots, L$: The set of services that process the data of $X^j$.
  $N$: Number of $(\mathcal{V}^j, \Phi^j)$ pairs that are selected for each subplan $\mathcal{L}^j$ (Optional
  parameter).
  $M$: Number of the multi-way join plans that will be pruned from the plan
  repository $\mathcal{R}$ (Optional parameter).
  $iter$: Maximum allowable number of iterations.
Outputs
  A multi-way join plan.

1: $\mathcal{R} \leftarrow \emptyset$; {Repository of multi-way join plans.}
2: Find a multi-way join plan $\mathcal{P}$ using the first phase of 2P-SO;
3: Add $\mathcal{P}$ to $\mathcal{R}$;
4: while there is at least one plan in $\mathcal{R}$ that is not marked as "visited", or the number
   of iterations is below $iter$ do
5:   for all multi-way join plans $\mathcal{P}$ in $\mathcal{R}$ that are not marked as "visited" do
6:     Let $\mathcal{L}^j$ be the bottleneck subplan of $\mathcal{P}$ with cost $\rho^j$;
7:     Select $N$ pairs of $(\mathcal{V}^j, \Phi^j)$ that are outputted by the algorithm in [18] when
       building $\mathcal{L}^j$;{Optional step. If this step is omitted, then all $(\mathcal{V}^j, \Phi^j)$ pairs
       that are returned by the algorithm in [18] are used.}
8:     for all selected pairs of $(\mathcal{V}^j, \Phi^j)$ do
9:       Create a new multi-way join plan $\mathcal{P}'$ from $\mathcal{P}$ using (i) the already found
         subplans $\mathcal{L}^i, i \neq j$, (ii) the subplan $\mathcal{L}^{j\,'}$ starting with $\mathcal{V}^j$ and followed (in
         any order) by the rest of the services in $\mathcal{W}^j - \Phi^j$ and (iii) the optimal
         subplan $\mathcal{C}'$ built by the algorithm in [18] using the join service $S^k_{join}$ and
         the services in $\mathcal{C}$ and $\Phi^j$, imposing the constraint that any service in $\mathcal{C}'$
         must be called after $S^k_{join}$;
10:      if the bottleneck cost of $\mathcal{P}'$ is less than $\rho^j$ then
11:        Add $\mathcal{P}'$ to $\mathcal{R}$;
12:      end if
13:      if none of the newly created multi-way join plans $\mathcal{P}'$ is added to the repos-
         itory then
14:        Plan $\mathcal{P}$ is marked as "visited";
15:      else
16:        Plan $\mathcal{P}$ is removed from $\mathcal{R}$;
17:      end if
18:    end for
19:  end for
20:  Prune from $\mathcal{R}$ the $M$ non "visited" multi-way join plans with the highest bot-
     tleneck costs;{Optional step.}
21: end while
22: return the multi-way join plan from $\mathcal{R}$ with the lowest bottleneck cost;
```

Fig. 11 The steps of A2P-SO

$(\mathcal{V}^j, \Phi^j)$ pairs that are returned by the algorithm in [18] when building a subplan $\mathcal{L}^j$ are utilized, in order to create new multi-way join plans. Those multi-way join plans are stored in a plan repository $\mathcal{R}$.

A2P-SO starts by building a multi-way join plan $\mathcal{P}$, as done in the first phase of the algorithm of Sect. 4. Let $\mathcal{L}^j$ be the bottleneck subplan of $\mathcal{P}$ created after the first phase of the algorithm. During the second phase, for every pair $(\mathcal{V}^j, \Phi^j)$ returned by the algorithm in [18] when building $\mathcal{L}^j$, a new multi-way join plan $\mathcal{P}^{'}$ is created as follows: instead of removing a service from $\mathcal{L}^j$ and appending it at the end of $\mathcal{C}$[12], the new algorithm creates a new multi-way join plan using (i) the subplans $\mathcal{L}^i, i \neq j$ of the multi-way join plan $\mathcal{P}$, (ii) the subplan $\mathcal{L}^{j^{'}}$ starting with $\mathcal{V}^j$ and followed (in any order) by the rest of the services in $\mathcal{W}^j - \Phi^j$ and (iii) the optimal subplan $\mathcal{C}^{'}$ built by the algorithm in [18] using the join service S^k_{join} and the services in $\mathcal{C}$ and Φ^j, imposing the constraint that any service in $\mathcal{C}^{'}$ must be called after S^k_{join}.

If the bottleneck cost of the new multi-way join plan $\mathcal{P}^{'}$ is less than the cost of $\mathcal{P}$, then the new plan is added to the repository $\mathcal{R}$. If none of the newly created multi-way join plans $\mathcal{P}^{'}$ is added to the repository, then plan $\mathcal{P}$ is marked as "visited". Otherwise, plan $\mathcal{P}$ is removed from $\mathcal{R}$. The second phase is repeated using the newly created multi-way join plans i.e., the plans of the repository that are not marked as "visited". The algorithm stops either when all plans in the repository are marked as "visited", or when a predefined number of iterations is reached. The complete algorithm is shown in Fig. 11.

In order to speed up the building process, we can use only N of the $(\mathcal{V}^j, \Phi^j)$ pairs returned by the algorithm in [18]. The N pairs can be selected either randomly or using a greedy approach, i.e., we can select the pairs for which the resulting $\mathcal{L}^{j^{'}}$ subplans have the lowest bottleneck costs; however, the pair $(\mathcal{V}^j, \Phi^j)$ that creates the lowest cost subplan $\mathcal{L}^{j^{'}}$ does not necessarily create the lowest cost multi-way join plan $\mathcal{P}^{'}$. Apart from that, after each iteration performed during the second phase, we can optionally reduce the search space (line 20 of Fig. 11).

References

1. Papazoglou MP, Traverso P (2007) Service-oriented computing: state of the art and research challenges. Computer 40:38–45
2. Oinn T, Greenwood M, Addis M, Alpdemir N, Ferris J, Glover K, Goble C, Goderis A, Hull D, Marvin D, Li P, Lord P, Pocock M, Senger M, Stevens R, Wipat A, Wroe C (2006) Taverna: lessons in creating a workflow environment for the life sciences. Concurr Comput Practice Experience 18(10):1067–1100
3. Taylor I, Shields M, Wang I (2004) Resource management for the Triana peer-to-peer services. In: Nabrzyski J, Schopf JM, Weglarz J (eds) Grid resource management. Kluwer Academic Publishers, Norwell, MA, pp 451–462
4. Deelman E, Singh G, Su M-H, Blythe J, Gil Y, Kesselman C, Mehta G, Vahi K, Berriman GB, Good J, Laity A, Jacob JC, Katz DS (2005) Pegasus: a framework for mapping complex scientific workflows onto distributed systems. Sci Progr J 13(3):219–237
5. Antonioletti M, Atkinson MP, Baxter R, Borley A, Hong NPC, Collins B, Hardman N, Hume AC, Knox A, Jackson M, Krause A, Laws S, Magowan J, Paton NW, Pearson D, Sugden T, Watson P,

[12] This is done in a way similar to the one employed during the second phase of the algorithm presented in Sect. 4

Westhead M (2005) The design and implementation of grid database services in OGSA-DAI. Concurr Pract Experience 17(2–4):357–376

6. Lynden S, Mukherjee A, Hume AC, Fernandes AAA, Paton NW, Sakellariou R, Watson P (2009) The design and implementation of ogsa-dqp: A service-based distributed query processor. Future Generation Computer Systems 25(3):224–236
7. Craddock T, Lord P, Harwood C, Wipat A (2006) E-science tools for the genomic scale characterisation of bacterial secreted proteins. All hands meeting, pp 788–795
8. Wang C, Chen M-S (1996) On the complexity of distributed query optimization. IEEE Trans Knowl Data Eng 8(4):650–662
9. Srivastava U, Munagala K, Widom J, Motwani R (2006) Query optimization over web services. In: Proceedings of the 32nd conference on very large databases (VLDB), pp 355–366
10. Van Den Hengel A, Dick A, Detmold H, Cichowski A, Hill R (2007) Finding camera overlap in large surveillance networks. In: Proceedings of the 8th Asian conference on computer vision, vol part I, pp 375–384
11. 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 (ICDE), pp 143–154
12. Lin W, Sun M-T, Poovendran R, Zhang Z (2010) Group event detection with a varying number of group members for video surveillance. IEEE Trans Circuits Syst Video Technol 20(8):1057–1067
13. Gibbons PB, Karp B, Ke Y, Nath S, Seshan S (2003) Irisnet: an architecture for a worldwide sensor web. IEEE Pervasive Computing 2(4):22–33
14. Kansal A, Nath S, Liu J, Zhao F (2007) Senseweb: an infrastructure for shared sensing. IEEE Multi-Media 14(4):8–13
15. Deshpande A, Hellerstein L (2008) Flow algorithms for parallel query optimization. In: Proceedings of the 24th international conference on data engineering (ICDE), pp 754–763
16. Babu S, Motwani R, Munagala K, Nishizawa I, Widom J (2004) Adaptive ordering of pipelined stream filters. In: Proceedings of the 2004 ACM SIGMOD international conference on management of data, pp 407–418
17. Viglas SD, Naughton JF, Burger J (2003) Maximizing the output rate of multi-way join queries over streaming information sources. In: Proceedings of the 29th international conference on very large data bases (VLDB), pp 285–296
18. Tsamoura E, Gounaris A, Manolopoulos Y (2011) Decentralized execution of linear workflows over web services. Future Gener Comput Syst 27(3):341–347
19. Ledlie J, Gardner P, Seltzer M (2007) Network coordinates in the wild. In: Proceedings of the 4th USENIX conference on networked systems design & implementation, p 22
20. Bianculli D, Ghezzi C (2007) Monitoring conversational web services. In: Proceedings of the 2nd international workshop on service oriented, software engineering, pp 15–21
21. Parker R, Rardin R (1984) Guaranteed performance heuristics for the bottleneck travelling salesman problem. Oper Res Lett 2(6):269–271
22. Culler D (2003) Planetlab: an open, community driven infrastructure for experimental planetary-scale services. In: USENIX symposium on internet technologies and systems
23. Kossmann D (2000) The state of the art in distributed query processing. ACM Comput Surv 32(4): 422–469
24. 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
25. Condon A, Deshpande A, Hellerstein L, Wu N (2009) Algorithms for distributional and adversarial pipelined filter ordering problems. ACM Trans Algorithm 5(2):24–34
26. 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 (PODS), pp 79–88
27. Ganguly S, Hasan W, Krishnamurthy R (1992) Query optimization for parallel execution. In: Proceedings of the 1992 ACM SIGMOD international conference on management of data, pp 9–18
28. Kossmann D, Stocker K (2000) Iterative dynamic programming: a new class of query optimization algorithms. ACM Trans Database Syst 25(1):43–82
29. Li J, Deshpande A, Khuller S (2009) Minimizing communication cost in distributed multi-query processing. In: Proceedings of the 21st international conference on data engineering (ICDE), pp 772–783