Available online at www.sciencedirect.com
SCIENCE DIRECT®
The Journal of Systems and Software 79 (2006) 480–495
The Journal of Systems and Software
www.elsevier.com/locate/jss

Efficient and adaptive discovery techniques of Web Services handling large data sets ☆

Christos Makris, Yannis Panagis, Evangelos Sakkopoulos *, Athanasios Tsakalidis

Department of Computer Engineering and Informatics, School of Engineering, University of Patras Rio, 26500 Patras, Greece
Research Academic Computer Technology Institute, Internet and Multimedia Technologies Research Unit 5, 61 Riga Feraiou Str. 26110 Patras, Greece

Received 16 April 2005; received in revised form 1 June 2005; accepted 2 June 2005
Available online 19 July 2005

Abstract

Attempts have been made concerning the search and finding of a Web Service based on keywords and descriptions. However, no work has been done concerning the efficient selection of the appropriate Web Service instance in terms of quality and performance factors at the moment of the Web Service consumption attempt. Such factors may include execution time and response time. The proposed approach adaptively selects the most efficient WS among possible different alternatives with real-time, optimized and countable factors-parameters. Implementation issues and case study experiments are presented along with the corresponding results. Additionally, an optimal selection algorithm for series of Web Services requests is proposed. Finally, conclusions and future steps are discussed.

Keywords: Dynamic discovery; Adaptive Web Services; UDDI

1. Introduction

Web Services (WS) provide a ubiquitously supported framework for application-to-application interaction, based on existing Web protocols and open XML standards. The Web Services framework is one of the newest members in the Web engineering area (Deshpande et al., 2002). It is divided into three areas: communication protocol, service description, and service discovery. Several specifications have been developed like SOAP (W3C, 2003), Web Services Description Language (WSDL) (W3C, 2004a) and Universal Description, Discovery and Integration (UDDI) (UDDI, 2004), correspondingly. The discovery part is particularly interesting as the demand for WS consumption is rising. A series of questions arise concerning the methods and procedures to discover the most suitable WS to use.

There is much hidden behind the behavior and use of WS. Web Services have been theoretically designed to support four types of behavior such as request-response, request-only, response-only and solicit-response. However, the vast majority of WS, as well as business logic processes, are of the request-response type and only a limited number follows the single way message transaction behavior. The present work deals with problems mainly occurring in the first and greatest category, where the response of a Web Service is time and cost sensitive to the end user. Intuitively, the most important factor for the end user has to do with the necessary time that

☆ This work is partially supported by the Karatheodory Research Program of the University of Patras, Greece.
* Corresponding author. Address: Department of Computer Engineering and Informatics, School of Engineering, University of Patras Rio, 26500 Patras, Greece. Tel.: +30 26 1096 0349; fax: +30 26 1096 0322.
E-mail addresses: makri@ceid.upatras.gr (C. Makris), panagis@ceid.upatras.gr (Y. Panagis), sakkopul@cti.gr (E. Sakkopoulos), tsak@cti.gr (A. Tsakalidis).

0164-1212/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2005.06.002

a Web Service takes to produce and deliver its results, that is the Web Service execution period. There are situations where Web Services have to deal with very large data or may need "long" execution time period for other reasons (low infrastructure availability) before returning results back to the customer/user. The Web Service execution duration varies depending on the current status of the infrastructure supporting it according to the available resources executing the Web Service. As a result the customer may asynchronously wait for the Web Service to respond so "long" as to even decide to request results from another available Web Service in a different location and with higher potential availability.

Such Web Services are particularly met in business environments where time and data intensive transactions are performed between customers and offered services/products. A typical example is found in the telecommunication business. A common telecommunication Web Service paradigm is the presentation of online analytic details for telephone calls. Carriers exist with millions of customers and billions of daily telephone call transactions. In such case, the Web Service has to patiently search for all the unbilled calls of a whole day's transactions to present analytically a single customer's telephone calls. Imagine now that this long-taking-to-respond Web Service may additionally be subject to extra delays. These delays depend on the availability and performance specifics of the carrier infrastructure resources. Consequently taking into consideration these specifics while selecting and binding to the Web Service is important for the effectiveness of the Web Services final choice.

2. Motivation

Our techniques and methods have been derived while studying the case of the greatest private nation-wide telecommunication carrier's network in Greece. In general, telecommunication carriers utilize gateways to provide fixed and mobile telephony services that have the capability to record call transactions either on proprietary files or directly on relational databases. (Implementation details are provided in the case study section following.) In particular, existing web and database applications and services implementations have served as the basis for this study. Moreover, the carrier utilizes the corporate network both for end-user internet service provision as well as for the needs of data interchange with its thousands of national business partners.

As a consequence, business partners as well as other associates have had the need to consume several kinds of Web Services that deliver various billing reports and exchange data concerning the *customer/call detailed records* (cdr). The cdr record is the footprint of a telecommunication service transaction and includes billing and communication information (caller, destination, duration, charge, etc.). Every single transaction may produce up to four (4) cdr records stored in a database, storing in this way huge amount of data progressively. Preliminary handling of such large data set is usually already performed in batch operations once per couple of days in order to facilitate compilation of billing reports. However, our initiative was to provide added value "advanced" Web Services that would allow associates and business partners to consume "online" updated information of a customers bill even for the unbilled cdr transactions not yet compiled.

Additionally, information about Web Service based consumption of customers transactions is also needed to the associates. The solution designed had to provide the necessary cdr transaction information to allow asynchronous detection and possible prevention of faked overcharging. Such overcharging can be performed manually by malicious users or automatically by computer viruses in cases of "hacked" or "snooped" telephone account PINs and passwords. Generally, in telephony services every customer is assigned one or more personal identification number and corresponding password for the authentication to the system just as the cases of card-based telephony. The telecommunication carriers (including internet service providers—ISP) need measures to protect their customers from unexpected fluctuations in their bills. As a result, a number of services have been developed and published to meet the needs of the business partners and associates in different points of presence (POP) at the corporate network. To provide quality of service (QoS) in these new added-value services, we had to maximize the throughput and execution time of each Web Service by efficiently discovering and binding to the most appropriate Web Service in order to maintain the sense of "online" response.

Some attempts have been made concerning the search and discovery of a Web Service based on keywords and descriptions of a WS. However, in the area of Web Service architecture and discovery and to the authors best knowledge, no prior work has been proposed concerning the "online" dynamic selection of WS at the moment of binding and consumption process. This work presents an effective and efficient dynamic selection of the appropriate Web Service instance in terms of quality and performance factors such as the execution and response time that the Web Service host can provide. Earlier work such as the *Object Management Group* (OMG) Trading Service, which was standardized in the mid-1990s and was based on research extending over the prior decade, supports dynamic properties, but it is both difficult to implement and it has been only partially adopted for reasons that led to the development and wide acceptance of Web Service technologies in first place. In this work, response time does not merely refer

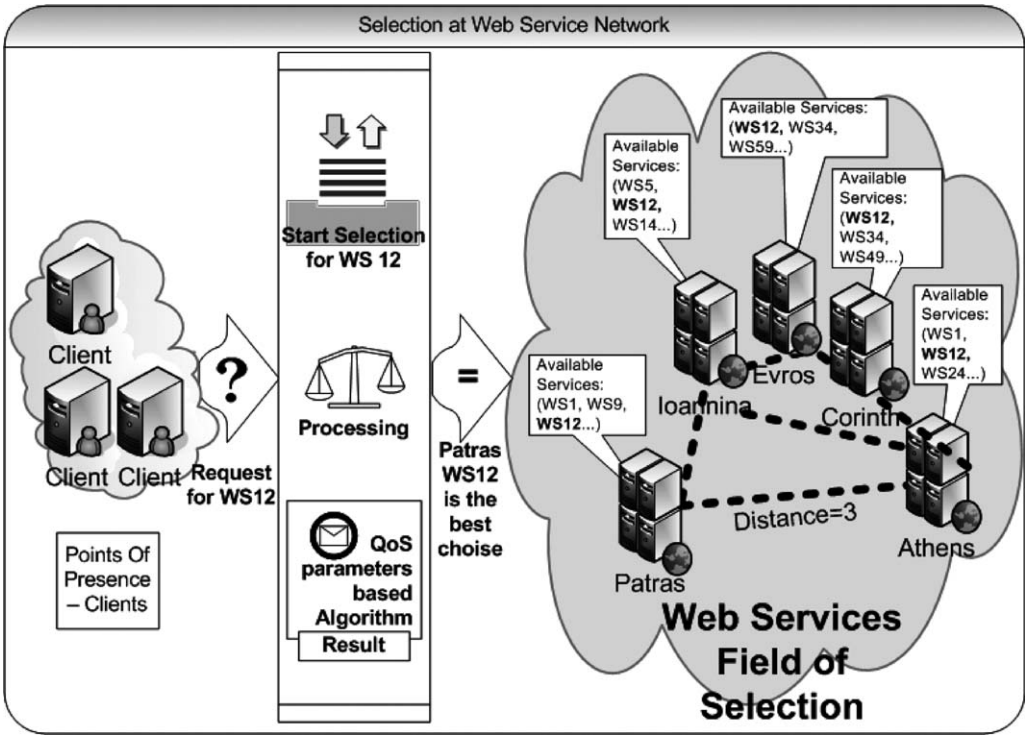

Fig. 1. General view of the Web Service selection problem.

to the minimization of Web Service message hand shaking but involves the overall duration of WS binding and consumption. The proposed approach adaptively selects the most efficient WS among different alternatives with optimized real-time countable factors-parameters. An early academic report of the authors technique for dynamic load balancing based on measurement of current service load appears in Makris et al. (2004), before its application on the telecommunication carrier, with no experimental results and without support for Web Service workflow operations. An abstract instance of the problem as a general view is depicted in Fig. 1.

The rest of the paper is organized as follows: Section 3 presents related work. In Section 4, we outline the algorithms and the required preparation steps, whereas the proposed efficient and adaptive selection algorithm is described in Section 5. Section 6 includes implementation issues and case study experiments and results. Furthermore, a step further that has to do with an optimal selection algorithm for a series of Web Services is presented in Section 7. Conclusions and future steps are included in Section 8.

3. Related work

3.1. Sharing data and consuming remote methods

Before proceeding to early and current trends on the Web Service Discovery techniques, a broader view of the background will unveil the different aspects and technologies involved. An important concept in modern software development is that of Service Oriented Architecture or SOA (W3C, 2004b). SOA is a, relatively new, attempt to face the old problems of sharing data and consuming remote methods. Typical scenarios, where the former actions appear, include businesses sharing information with customers, departments sharing data with other departments and end-user applications collaborating with other end-user applications. All these cases are also treated by the SOA solution.

These tasks have occurred since the early years of computer intercommunication and several methodologies have previously been presented resulting to the introduction of the so-called distributed application development. Distributed application development focuses in engineering methods to get data from one host to another. In fact, there have been many technologies for building applications that can send data back and forth; *CORBA* (Common Object Request Broker Architecture), *RMI* (Remote Method Invocation), and *DCOM* (Distributed Component Object Model) just to name a few.

All of the aforementioned approaches presented significant difficulties to catch up and gain wide adoption from the development community. A typical example is DCOM, the distributed counterpart of Microsoft COM (Component Object Model). DCOM usage is essentially also limited to Windows platforms, although it is feasible to develop cross-platform

solutions. Common Object Request Broker Architecture, or CORBA, is also a distributed technology designed to facilitate development over heterogeneous environments. Nevertheless, CORBA application development is particularly involved, a reason for CORBA not to become really popular.

Each of the above has failed to succeed due to several reasons including, very complex implementation and infrastructure requirements, complex and/or proprietary communication protocols, lack of support of large software companies and platform restrictions. Therefore, the development community was led to devise the Web Services architecture.

3.2. Web Service Discovery techniques

Though the field of Web Service Discovery is rather new, even newer than Web Services themselves, much work has been lately devoted to the area. The effort in the bulk of the approaches is to enhance the Discovery mechanisms in order to overcome the inadequacy of the standard, keyword-based matching, where often the user cannot discover the Web Service she wants.

In this vein Sajjanhar et al. (2004) have proposed representation of Service Descriptions as texts and by mapping the generated text-vectors to a low-dimensional feature space they are able to observe better Web Service retrieval. A unified solution has been proposed by Zhuge and Liu (2004), who presented an SQLlike flexible query language to support the flexible retrieval of services. They have built upon use of the *Service Grid* (Zhuge, 2002a,b,c), which organizes services in a normalized and orthogonal multidimensional service space.

An important trend in Web Service Discovery is to consider network nodes as peers, sharing their available Web Services and being able to query other nodes. A number of papers have been recently proposed that are mainly based on using Chord (Stoica et al., 2003) as the underlying network topology. Their main advantage is that they can inherently support load balancing. In the work of Schmidt and Parashar (2004), a single WS is considered as a multidimensional point. They also use a space-filling curve, a transformation which injectively maps points in higher dimensions to numbers, the service IDs. The latter are distributed among the network peers according to Chord principles.

Li et al. (2004) also use keyword mapping to a Chord of Web Service Peers. Their mapping though, is achieved using distributed hashing; the service descriptions are hashed and the hashed IDs are then distributed to the Chord network, giving thus what they call XChord. A Web Service query starting at a peer node, is also decomposed into keywords which are subsequently sought for using the Chord searching principle. XChord is proved more stable, load balanced and less space consuming in comparison to standard UDDI, according to the conducted experiments. In Rao and Su (2004), the Chord P2P protocol is also utilized as overlay including service peers. The architecture introduced is called P2P-based Web Service Discovery (PWSD). Web Service descriptions as well as queries are hashed and routed in the Chord network. A different approach called NIPPERS is presented in Makris et al. (2005). This novel technique provides improved results in searching and managing WS compared to the popular DHT overlay network Chord. Cross-disciplinary decentralized solutions have also been proposed incorporating ontologies. In Schlosser et al. (2002), a system is presented using Peer-to-Peer (P2P) technologies and ontologies to publish and search for Web Services descriptions.

Significant contribution to Web Service Discovery has been made from the semantics community. Essentially the efforts are focused on modeling Web Services as ontologies and then perform semantic matching. A typical framework in this direction is that of Paolucci et al. (2002) that augments WSDL and UDDI in order to perform semantic matching. Service Profiles are generated that encapsulate all the functional characteristics of a Web Service, and descriptions are generated using DAML-S (Burstein et al., 2002). A recent development (Sycara, 2004) was the development of a new semantic language, namely OWL-S, that allows semantic annotations and semantic discovery to be integrated in the UDDI and WSDL. A similar matching approach is followed by Sivashanmugam et al. (2003). Moreau et al. (2002), essentially transform Web Services into XML Schemas and structurally match Web Service queries against those schemas. Finally, Overhage and Thomas (2003) introduces semantic descriptions in a special section of Web Service descriptions, the so called ''blue pages'' section. Zhang et al. (2004) presented the approach of WS-Net, which substitutes current WSDL structures. This technique takes advantage of the semantics of Colored Petri-net with the style and understandability of object-oriented concepts. It helps to enhance the reliability of web-services oriented applications. However, transformation of WSDL specifications into WS-Net is neither trivial nor automated.

One more interesting solution is the Woogle engine (Dong et al., 2004). Woogle faces the problem of approximate retrieval using clustering techniques. Clustering is formed on the basis of association rules' formation among the terms constituting operation names. The main intuition behind this approach is that, when all the parameters inside a set frequently co-occur, then they may correspond to similar functionalities.

The matching procedure in all the above cases, with the exception of Paolucci et al. (2002), does not take into

account concerns such as the response time that was highlighted as of primary concern in the previous section. However, in a real-world invocation environment, aspects such as response time are paramount. Characteristics such as execution time are met under the term, Quality of Web Service (QoWS) Characteristics. Ran (2003) presented some more desirable characteristics that define QoWS. A similar attempt was made by Ouzzani and Bouguettaya (2004). Their paper (Ouzzani and Bouguettaya, 2004) was the first to address the issue of Web Service Discovery that meets certain QoWS constraints. Although, the selection details are somewhat unclear, they present an integrated framework to QoWS Discovery.

Additionally, Yu and Lin (2005) present an efficient algorithm to select the optimum execution plan when a Web Service workflow is executed. They treat cases where the interest is focused on execution plans comprising of a sequence of k consecutively executed Web Services $\{S_1, S_2, \ldots, S_k\}$ or a "pipeline" as it is phrased. The selection of an execution sequence with total execution time below a threshold R is transformed into a Multiple Choice Knapsack (MCK). Yu and Lin (2004b), provide a dynamic programming algorithm and variations to solve MCK, and as a result to efficiently select the most appropriate service sequence. Furthermore, Yu and Lin (2004a, 2005) present a mechanism named *QoS Broker for Web Services*. It is an intermediate between the UDDI registry and the end user(s) that performs all exhaustive tasks to locate and ensure the delivery of the most qualified Web Services.

A generalized perspective of the challenging task to devise a scoring function that combines the QoWS criteria in a righteous manner is presented in Liu et al. (2004). Their aim is to produce a single score for each Web Service or execution plan by representing each WS as an m-dimensional vector, where m is the number of parameters taken into account.

An extensive and comparative survey of Web Service Discovery mechanisms is also presented in Garofalakis et al. (2004).

4. Algorithmic roadmap

In this work an efficient and adaptive algorithm is proposed that performs selection among similar Web Services located at different infrastructures. We deal with the existence of a number of related services that offer similar functions such that any of them can fulfill what the consuming business partner or associate seeks. This is broadly met among the enterprises that collaborate with a specific range of partners in order to perform a certain series of operations, which are consequently semantically correlated. As a result, partners develop programmatic interfaces to discover and consume Web Services and tend to perform only narrowed queries for Web Services with similar functionality. Moreover, searches are conducted within a specific range of partner-Web Service provider. In such cases, our aim is to provide a more intelligent discovery procedure for dynamic selection of the most appropriate Web Service, than the naive "round robin selection". The latter would entail polling all available WS instances for their performance characteristics and then simply choose a WS that maximizes or minimizes a specific metric.

The algorithm includes two functional components that compose the final step of the Web Service selection. The overall logic of the algorithm is depicted in Fig. 2. In this figure the interaction between components is presented.

Before proceeding to the analysis of each functional component, an outline of the algorithm is presented. This presentation serves as a roadmap for the reader and the necessary details may be found in the subsections following.

(1) *Contour Selection:* At first the proposed algorithm takes into consideration two parameters and based on them, it performs an efficient selection of the best candidate Web Services. In Fig. 2, this step is depicted in Fig. 2a. The parameters include:
   - The network distance; it is the mean network latency between the client requester and the Web Service.
   - The number of other distinct Web Services, functionally related to the Web Service in terms of business environment; the parameter contributes to narrowing further the number of the candidate Web Services.

(2) As a second component, the *adaptive selection* process is proposed. It is based on online quality ratings of a Web Service (QoWS ratings). These are measures, i.e., countable factors such as available memory, etc., concerning the quality and the availability conditions of the infrastructure implementing the WS at the very execution moment. A

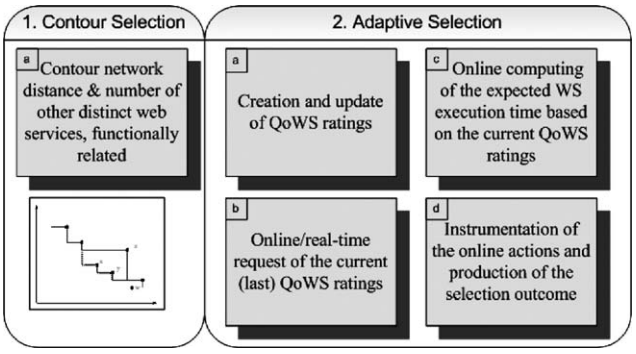

Fig. 2. Algorithmic roadmap.

service's average execution time is observed over a number of executions over time. At this point to assure the maximization of quality in the chosen Web Service, the fast corporate network is utilized to collect up-to-date WS measurements and deduce the estimation of the expected QoWS characteristics based on previously recorded WS execution schemes-profiles. The alternative of scheduling regular dynamic updates of UDDI QoWS properties can be utilized in environments, where corporate network traffic is already high.

This process includes one preparatory and three online/real-time actions:

(a) Creation and update of QoWS ratings; this task is a service that maintains WS history profiles. It is a preparatory action that records the QoWS ratings as well as the corresponding execution time of the WS and stores them as a history profile (hereafter called WS execution scheme). In particular it stores the *duration* that a WS took to process a request under specific QoWS ratings as well as the ratings themselves. This action serves as an add-in service that takes place prior to the selection procedure.This preparatory action maintains the WS execution schemes. Details are included in Section 5.2.1.
(b) Online/real-time request of the current (last) QoWS ratings; this action implements an agent service that returns the measures of the point of WS presence at hand, which are included in the WS execution scheme. This online action returns the current value of the countable factors in the WS POP at hand. It acts as a part of the main online action (following in Fig. 2d).
(c) Online computing of the expected WS execution time based on the current QoWS ratings; this action finds the nearest and therefore expected response time in the WS's execution scheme. It performs matching based on the Euclidean distance between the QoWS ratings in the WS execution scheme.This online action returns the *expected WS execution time* based on the values of several countable factors available in the WS POP. It acts as a part of the main online action (description in Fig. 2d).
(d) Instrumentation of the online actions and production of the selection outcome; this is the last online action that controls the previous two ones and performs the main adaptive selection steps of the algorithm proposed. It returns the final selected Web Service instance to be used. Detailed description of the action may be found in Section 5.2.2.

5. Efficient Web Service selection

5.1. A first selection step based on common computational geometry observations

The performance evaluation for each web-service-providing node is a computationally demanding process and thus we need to apply it to as few nodes as possible. Therefore, we use a pruning heuristic that allows us to rule out certain nodes as a first step. This heuristic is called contour selection. Before we proceed with the description, we provide some useful notions (Fig. 3).

Let S, denote the set of all candidate nodes such that each node provides a set of Web Services that we are interested in consuming. For an arbitrary element $s_i \in S$, we consider two parameters; its *network distance* from the query submitting node (any Point of Presence), d_i; and the number f_i of the *distinct Web Services* it provides. The choice of distance is based on the need to exclude at first, nodes having large distance from our entry point (any Point of Presence), thus the execution time plus network latency may potentially cause total execution time to be unnecessarily high. The choice of the number of provided Web Services is made by following the intuition that if a specific node provides a significant number of Web Services, then the probability to serve a large subset of the required set of Web Services, within the same node, increases, especially taking into consideration that consumers operate within a certain business framework (e.g., Web Services for tourism—seeking tour operator, booking boat tickets, reserving hotel room, etc.). The contour selection heuristic, selects nodes being maximal with respect to both network distance and number of functions.

More typically, each s_i, can be uniquely represented as a point $(-d_i, f_i)$ on the xy–plane, where the distance corresponds to the x-coordinate and the number of Web Services, to the y-coordinate. A point $s_i \in S \subseteq \mathscr{R}^2$, $s_i(x_i, y_i)$ is called *maximal* if there is no other

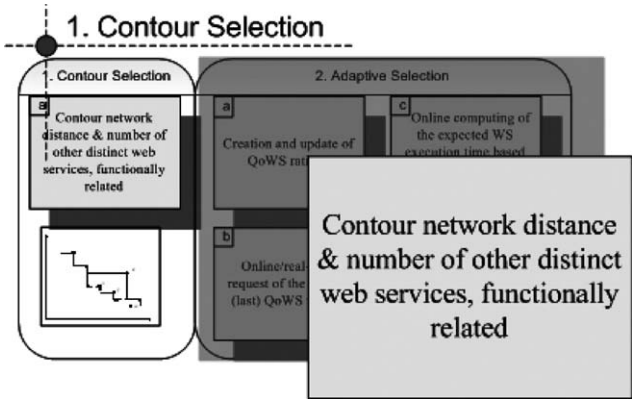

Fig. 3. The first step of the roadmap.

point $s_j(x_j, y_j)$ such that both $x_j \geqslant x_i$ and $y_j \geqslant y_i$. It can be easily verified that maximal points form a staircase in the plane, which from now and on we call *the contour* (see Fig. 5). The set $M \subseteq S$ of maximal points corresponds to servers that cannot be overruled in both the number of services and the shortest distance by any other server providing a specific service. Any of these points (servers) could lead potentially to a good choice regarding the service asked.

Given a set S of points on a plane, the subset M of the maximal points, can be computed with the algorithm CONTOUR_CONSTRUCTION. This algorithm, shown in Fig. 4, uses a balanced binary search tree T, to store points in M according to their x coordinate. We postulate that T is leaf-oriented, which means that points are only stored in the tree leaves and internal nodes only contain routing information. For more extensive information on the notion of leaf oriented trees, one can consult the textbook of Mehlhorn (1984). Each leaf v, apart from the x coordinate of the point p it stores, also keeps a pointer to p. We further assume that each tree-leaf v, contains pointers $v.left$ and $v.right$ to its left and right siblings, respectively. The algorithm CONTOUR_CONSTRUCTION presented in Fig. 4 constructs a contour representation given a set of point in the plane.

```
CONTOUR_CONSTRUCTION(x)
Input: A set of points S
Output: The set of maximal points M
1.  Sort points in S according to their x−coordinates
(* Let s_1, s_2, ..., s_n denote the sorted order *)
2.  for i = 1 to n
(* Current point is s_i(x_i, y_i) *)
3.     search T with x_i;
       (* Search will terminate at a leaf v storing point p(x, y)
       and having x ≤ x_i. *)
4.     cur← v;
5.     if (v.y < y_i) insert(T, x_i);
6.     while (v.y < y_i) do
7.        cur←v.pred;
8.        delete(T, v);
9.        v ← cur;
10.    end-while
11. end-for
```

Fig. 4. The algorithm for contour construction.

The process taking place during the contour construction is best illustrated in Fig. 5. Fig. 5 depicts the contour (solid line) for the point set $S = \{(-13,2), (-11,13), (-10,5), (-9,11), (-8,3), (-7,6), (-4,5), (-3,7)\}$. After having processed the first seven points, the insertion of $s_8(-3,7)$ in the tree T, is preceded by a search (line 3), which ends at the leaf storing -4. Next we store s_8 and search to the left of it in order to eliminate any points that the newly inserted one (s_8) dominates (lines 6–10). This leads to the deletion of $(-7,5)$ and $(-4,5)$ from T. This case is shown in Fig. 5b with the deleted leaves being shadowed.

An upper bound on the complexity to build the contour with the algorithm CONTOUR_CONSTRUCTION can be derived as follows. Clearly searches and insertions in an n-node balanced binary search tree take $O(\log n)$ time, for a total of $O(n \log n)$ total time. However, after the insertion of a point s_i in line 5, we might have to scan the tree leaves and delete any points stored at the leaves, whose x (and y) coordinates are dominated by s_i. We may have to do up to $O(n)$ such operations after a single insertion, for a cost of $O(n \log n)$. Nevertheless, the total cost for all executions of lines 6–10 is not $O(n^2 \log n)$. Suppose that the operations in lines 6–10, at the ith point insertion, delete $k_i \geqslant 0$ leaves, $i = 1, \ldots, n$. Hence, at the ith iteration the cost is $O(k_i \log n)$. Notice however that $\sum_{i=1}^{n} k_i = n$ since each point can be deleted at most once, for when a point is deleted it is not inserted again. Thus the total cost of the execution of lines 6–10, over

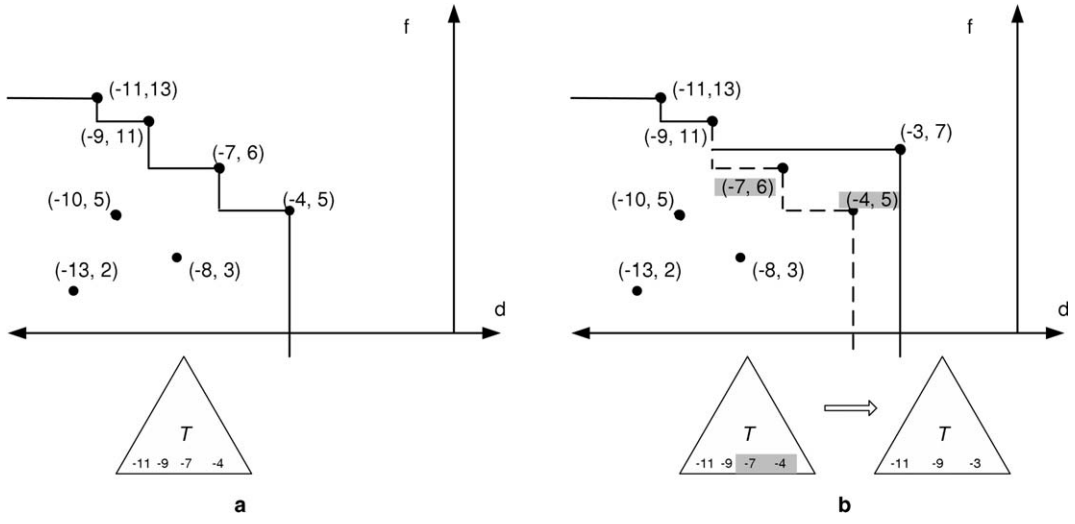

Fig. 5. The contour of $S = \{(-13,2), (-11,13), (-10,5), (-9,11), (-8,3), (-7,6), (-4,5), (-3,7)\}$. (a) The contour after the insertion of the first seven points. (b) Update to the contour after the insertion of $s_8(-3,7)$. Leaves deleted from T are shadowed. The new contour is shown with solid line, and the old with dashed line. The corresponding snapshots of T are shown in both cases.

all possible i is $\sum_{i=1}^{n} O(k_i \log n) = O(n \log n)$. The above discussion leads to the following lemma.

Lemma 1. CONTOUR_CONSTRUCTION *for n points is completed in* $O(n \log n)$ *time.*

5.2. Adaptive selection based on online QoWS ratings

Since the contour selection has been completed successfully, there is a list of more than one Web Service candidates for the final selection. In the following sections we will describe how to optimize the matching of registered Web Services based on the online rating feedback. The next step in the proposed selection method is the adaptive selection function. At first a preparatory action is presented, where the maintenance of Web Services' history profile is performed. In the sequel the online actions are presented in detail and the adaptive selection mechanism description is concluded.

5.2.1. Preparatory action: maintaining WSs history profile

The first preparatory action of the adaptive selection algorithm records the WSs' history profile (step (a) in Fig. 6). Every Web Service of interest is registered with a QoWS execution scheme (that is the WS history profile) that includes details of different execution plans. The execution plans include details (QoWS ratings) about the Web Service executed in a specific environment (these are countable measures of quality and availability of infrastructure). Additionally, every WSs' execution needs some time to be performed that is called the *execution time*. The most decisive countable factors that may prolong the execution time can be the percentage of mean *cpu usage*, the percentage of mean *memory usage*, the mean *number of processes* and the mean *file transactions* (over the last 30 min period). One may also take into consideration other real-time countable factors (see Microsoft, 2004a,b, for a complete list). Including to more or to less extent recorded factors to the execution plan does not affect the behavior of the selection algorithm since recording is a preliminary action.

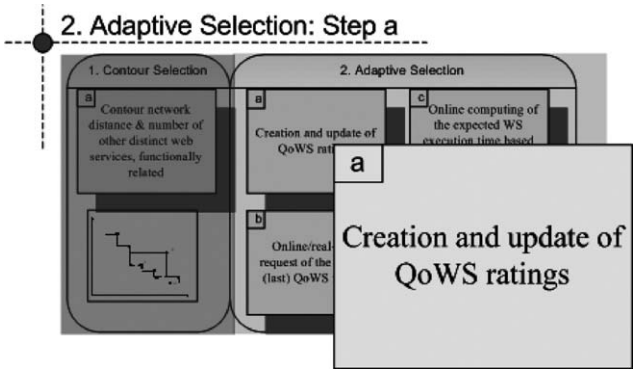

Fig. 6. Step 3 of the roadmap.

As a consequence the QoWS execution scheme includes a list of different scenarios that describe the execution time as a function of some recorded quality and infrastructure availability factors. In case of possible infrastructure change (due to hardware upgrade for example), the QoWS execution scheme should be recompiled.

The recorded parameters consist of counters and measures at system, service and application-specific instance level. There are some generic measures such as CPU, memory, disk I/O utilization and some which are level specific such as concurrent threads of the same application in the system (application level), transaction requests to other WSs (service level) and others. The information for QoWS execution schemes of all WSs in the network is replicated into a database together with the catalog of available WS.

5.2.2. Online actions: instrumentation of the adaptive selection algorithm

After the preparatory first step (step (a)) of the adaptive selection algorithm that involves the recording of QoWS schemes (history) of the Web Services of interest, we present the online actions of the adaptive selection algorithm that chooses the WS with better responses among the candidates (the steps involved are depicted in Fig. 7). The players of the online actions are:

(1) the request agent service of the current QoWS ratings,
(2) the computation algorithm of the expected WS execution time based on the results of the fetched ratings and
(3) the managing algorithm that performs selection.

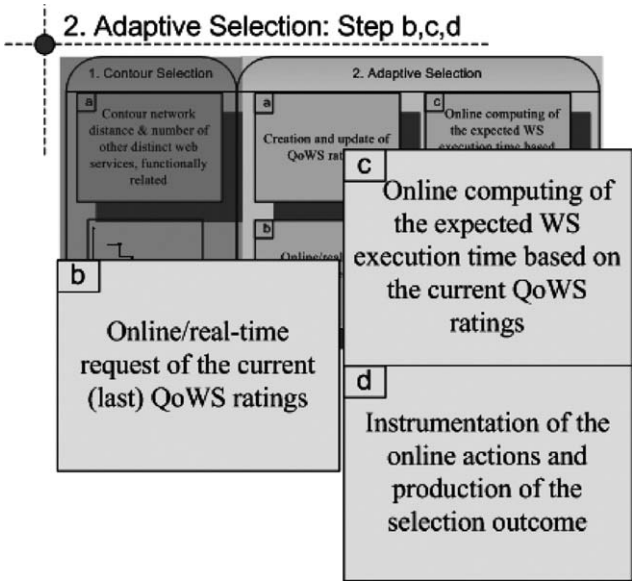

Fig. 7. Online selection steps.

We have built an agent service that can communicate with every host and retrieve the current (last) measure information of the execution time parameters function. Agent services of this kind are based on the performance utilities provided by the operating system and the specific applications wrapped by Web Services instances (i.e., SQL server based WS can get QoWS ratings). Implementation details of the agent service may be found in the next section. The agent service returns the current values for the chosen parameters of the infrastructure implementing the WS as a result. An example of recorded QoWS ratings is displayed in Table 1.

Table 1 presents an example of a WS execution scheme. It comprises the WS identification number, the recorded execution time in seconds, a date and time id and several system, application and service level countable measures. As already described, the execution characteristics for this WS and at several discrete time instances are recorded.

Having maintained such history profiles the current QoWS ratings are matched with an expected execution time based on the history profile of the infrastructure. The matching is performed by computing the Euclidean distance between the current QoWS ratings vector and the stored profile vectors. More concretely, consider the profile vector $h = [h_1, h_2, \ldots, h_p]$, consisting of p stored parameters and a vector $c = [c_1, c_2, \ldots, c_p]$ of current service parameters. The Euclidean distance L_2 between h and c is

$$L_2 = \sqrt{\sum_{i=1}^{p} (c_i - h_i)^2}. \quad (1)$$

The vector $h_{\min}$ that minimizes L_2 provides an estimation for the current execution time. Possible utilization of L_1 or L_∞ distance metrics would provide similar results.

The mechanism, which manages the above actions, performs the main adaptive WS selection. Firstly, the pruning step of Section 5.1 is performed. This step possibly reduces the working space of Web Service selection candidates to the services belonging to the contour. Let this set be W'. For a specific Web Service in W', its past execution times are recorded and all execution times are sorted in ascending order. We use two arrays ET and r for storing, estimated execution times for the candidate Web Services and *real* execution times at the moment of selection, respectively. In order to derive the initial values for the entries of ET we can use two strategies depending on the network load:

(1) Let the QoWS agent ask all nodes concurrently for their current workload, collect responses, match against stored profiles and set the estimated time according to the execution time of the nearest profile with respect to the L_2 metric.
(2) Set $ET[i]$ to the average of all execution times recorded in the service profile.

Let *min* denote the minimum execution time. Initially, *min* takes the minimum value stored in ET. The entries of r are computed online in selection time by asking each node the appropriate parameters and using again the L_2 metric. We note again that multicast queries could be utilized to ask clusters of nodes existing in local area network for faster responses.

For each i, we have computed the network latency $L[i]$ that corresponds to the communication time that is needed to contact server i. Let $\mathscr{L}$ also denote the sum of all network latencies.

The selection relies on two facts. Firstly, if the total network latency $\mathscr{L}$, required to ask every node, is smaller than the current minimum execution estimation, then we can recompute all execution time estimations in the hope that we may come up with a new minimum. If $\mathscr{L}$ is larger than the minimum expected execution time, we cannot afford asking all nodes hoping to a more accurate estimation, because we will spend more time in asking than in following the best execution scenario. The algorithm must define, thus, a stopping criterion; it keeps searching for up-to-date estimated times as long as the latency paid so far plus the latency in asking the next node is smaller than the minimum time encoun-

Table 1
A simple WS execution scheme paradigm

Recorded execution time per WS instance			System level			Application level		Service level
WS id	Execution time (s)	Date–time id	Number of concurrent processes	% CPU max process	Mean % of available memory	Number of concurrent threads of same application	Number of disk utilization, reads/writes $\times 10^3$	Number of other WS transactions
12	61	0	2	5	82	1	≈200	0
12	53	1	1	6	91	1	≈180	0
12	244	2	13	25	56	2	≈1000	0
12	351	3	41	80	20	10	≈1500	0
12	157	4	9	43	74	8	≈800	0

```
ADAPTIVE_SELECTION(WSD)
Input: A Web Service description request WS_D and a set of candidates W'
Output: min, the WS with best expected execution time
1.  if (L < min)
    (* Minimum execution time larger than the cost to ask everyone *)
2.    compute new estimations for each candidate and store them to ET;
3.    min ←the new minimum execution time estimate;
4.  else
5.    j←1;
6.    compute r[1];
7.    min ← r[1];
8.  end-if
9.  while (∑_{i=1}^{j} L[i] + L[j+1] < min)
10.   compute r[j+1];
11.   if (r[j+1] < min)
12.     min ← r[j+1];
13. end-while
14. return min
```

Fig. 8. Online selection algorithm.

tered so far. Any steps further will not be of any use, even if a smaller actual time is found, since we have already spent more time in seeking for an answer than in executing the minimum execution time Web Service. The actual selection algorithm ADAPTIVE_SELECTION is shown in Fig. 8. Computing the most up-to-date execution estimate is presented in Fig. 8 with the command "compute" (lines 6 and 10).

Note at this point that our algorithm is temporarily biased toward certain Web Service providing sites, for they are optimal with respect to certain criteria. However, this situation changes during the execution should certain nodes receive more traffic and thus respond with larger execution times.

5.3. Performance evaluation

The rationale for using the contour selection heuristic is twofold: (i) the set of candidate Web Service nodes is reduced significantly, in the average case, (ii) it is guaranteed that the total communication cost due to network latency is reduced.

In the sequel, we outline a performance evaluation analysis, which aims to clarify the intuition behind the contour selection. This analysis is also experimentally validated, in the next section. Hence, let m be the number of the candidate Web Service nodes; let m' be the number of the nodes that belong to the contour; let t be the number of entries in the array storing the current profile vectors of the specific WS and let l be the maximum communication cost due to network latency. Then the time complexity without the contour selection heuristic will be: $O(m \log m)$ {for sorting the candidates using, e.g., mergesort (see Cormen et al., 2001; Mehlhorn, 1984)} + $O(mt)$ {for accessing the cells of the array} + $O(l)$, while the time complexity when applying the contour selection logic will be: $O(m \log m)$ {for computing the contour} + $O(m' \log m')$ {for sorting the candidates} + $O(m't)$ {for accessing the cells of the array} + $O(l')$.

Since l' is always at most l it follows that the contour selection preprocessing is surely justified when $mt > m' \log m' + m't$. However one should note that: (i) in the general case l' is strictly less than l and (ii) the dominant term in the contour selection is the communication cost due to network latency and since the contour selection logic guarantees that this cost is reduced, it can be expected that the total time complexity (being dominated by the communication cost) will be reduced.

6. Implementation and case study results

The technological environment used for the implementation of the mechanisms proposed as well as the evaluation of them is the MS .NET framework version 1.1[1] and the C# programming language (Microsoft, 2005). The experimental computations ran on Windows 2003 Intel Pentium IV (3 GHz) servers with 1 GB up to 1.5 GB RAM. The MS SQL 2000 with sp3 database management systems were used. The main reasons, which have driven the specific implementation choices, were to promote compatibility with a strategic choice for technological platform as well as an attempt to facilitate interoperation with existing services and infrastructure.

A telecommunication carrier's corporate network is a well-known and balanced network where the mean network latency is far less than the mean execution time of a Web Service handling large date sets. Generally, such is the case of most business processes in large corporate or other networks using Web Services that have to deal with large amounts of data before computing their response. Results that highlight this are presented in the following. Nevertheless, comparative experimental results are also presented for cases where network latency is close to execution time, which show that our proposed solution works efficiently in both cases. Finally, to evaluate the particular impact of the contour selection algorithmic step, comparative results are also presented.

6.1. Setting up the experiment

The methodology of the experiments is described in the following list.

(1) The QoWS execution scheme per Web Service was recorded under several different environmental parameters for a list of twelve (12) different Web Services in sixty six (66) different instances each—each instance represents different infrastructure conditions/states.

[1] http://www.microsoft.com/net/

(2) WS are independent of each other. Composite Web Services were acceptable as long as they did not utilize other participating WS.
(3) Distinct QoWS characteristics are participating in the execution schemes. Without affecting generality as more QoWS maybe utilized, we assumed five (5) QoWS parameters being recorded: CPU utilization, memory utilization, disk I/O operations, number of application threads and number of system processes running on the infrastructure. All parameters were normalized to facilitate computations: QoWS $\in [0,1]$.
(4) The execution duration as well as the network latency values range: $ET \in [10,3600]$, $nt \in [1,60]$.
(5) The same WS list was imported into a UDDI registry server implementation (UDDI service on Windows 2003 Server (Microsoft, 2003)). All instances of WS are recognized based on their unique Web Service key which is used to return to the consumer the appropriate WS description reference.
(6) Random Web Service consumption requests have been chosen. Making random service requests does not influence the selection outcome: we would obtain the same results ever if the requests were specific. Furthermore, random request generation ensures that our algorithm is not influenced by the specific business process, i.e., by the request context. Both the UDDI implementation and the proposed algorithm returned candidate selection WS instances of the same WS functionality under the same infrastructure conditions/states.

Next we present comparative diagrams which depict the results using two axes; the *y*-axis depicts execution time of the WS selected by each method, and the *x*-axis depicts the queries for different WS of random choice. The diagrams in Figs. 9 and 10 present the UDDI return results versus the proposed algorithm results. In order to facilitate the discussion of experimental results, we have aggregated them into four categories. Results in each category have similar execution attributes. Results in different categories have different ranges in execution time and/or network delays to access a WS. The different sets are shown in Table 2.

6.2. Evaluation of the measurements

In Figs. 9 and 10 the dotted line depicts the execution time of the WS selected by the proposed algorithm versus the WS's ET based on a UDDI selection. As the dotted line is at most cases below the UDDI line, it appears that the proposed selection generally results in better total execution times. In fact the exact average distance between the result-lines of the four sets may be found in Table 2 above. The experimental results indicate a WS response time improvement of 25.29%, 42.71%, 57.89%, 66.47% correspondingly at each set. The average gain of all sets is 48.09%. The experiments revealed that the proposed algorithm returns improving results as the execution and network time of a WS increase.

The results in Fig. 11 strengthen the hypothesis that the proposed algorithm provides response time gain through online WS selection. The proposed algorithm provides efficiently maximized WS performance selections as well as it confirms the availability of the WS.

6.3. Evaluation of contour algorithm impact

To demonstrate the impact of the initial selection algorithm using the contour (see step 1 in Fig. 2), we evaluate the gain using comparative results in different cases. In Fig. 12 selection is performed on Web Services of WS Set 1 and 2. Fig. 12 presents the behavior of total execution time after performing the online selection pro-

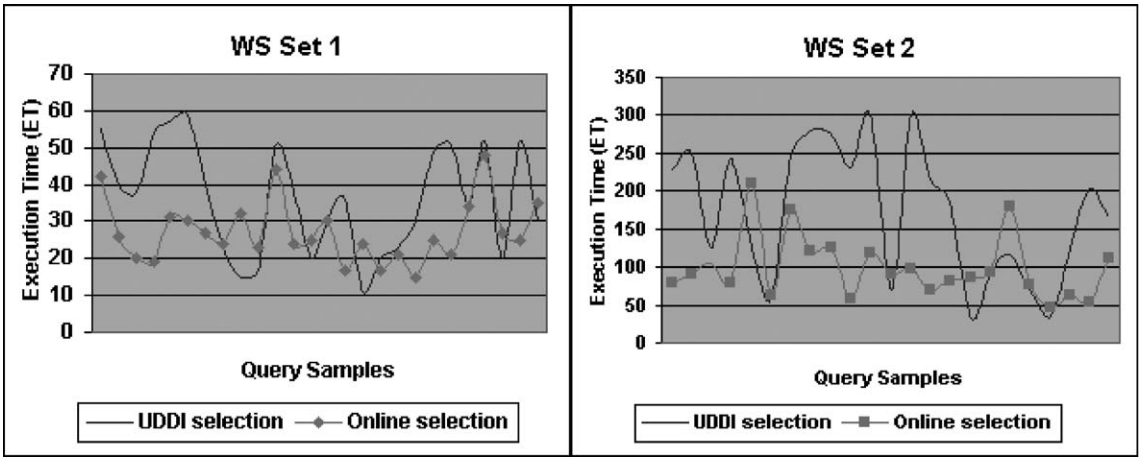

Fig. 9. Experimental results with the two shorter sets of execution and network latency durations.

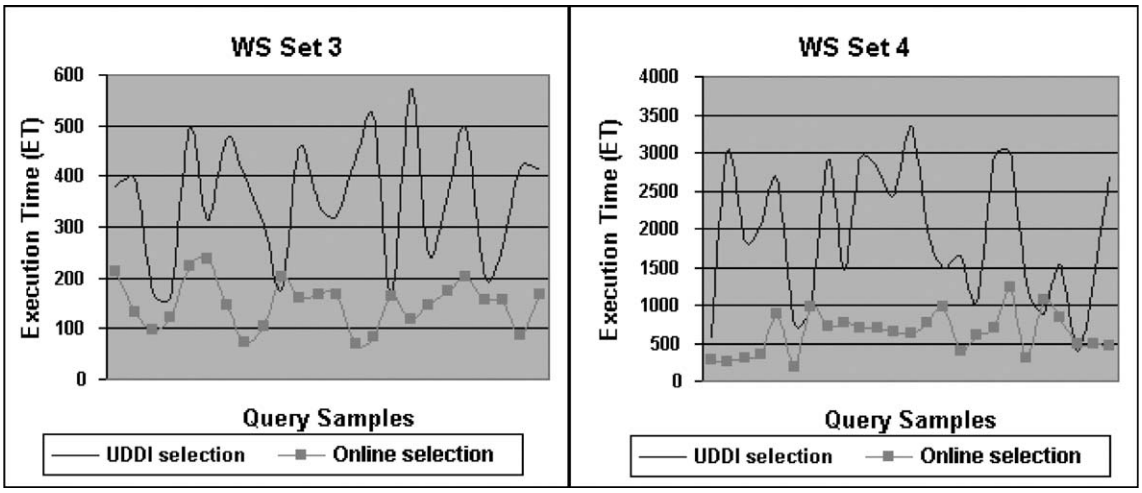

Fig. 10. Experimental results with the two shorter sets of execution and network latency durations.

Table 2
Experimental results grouped into sets

Result set ID	Execution time (s)	Network time (s)	Average difference of UDDI vs online selection (s)
WS Set 1	10 up to 60	1 up to 30	+5.29
WS Set 2	30 up to 320	1 up to 30	+33.42
WS Set 3	60 up to 600	1 up to 30	+69.18
WS Set 4	180 up to 3600	10 up to 60	+399.85

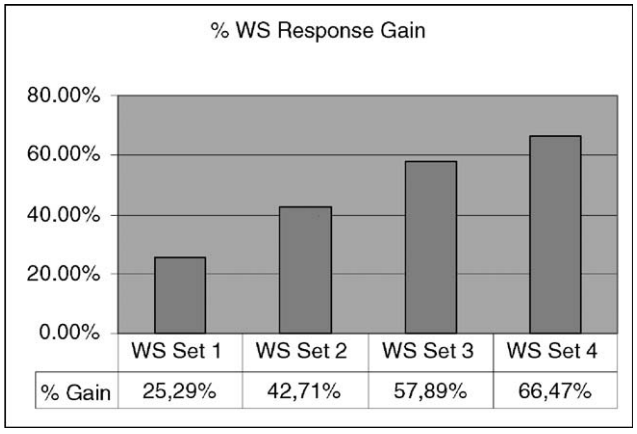

Fig. 11. WS response time gain of the experimental result sets.

cedure of Section 5.2 with and without the initial pruning step of Section 5.1.

The comparison has shown that contour based selection provides an initial threshold to reduce the search space to those WS that have minimized network distance (response) and maximized functionality. In particular in WS Set 1 where the network distance is close to the execution time the gain is 8.37%. In the right hand part of Fig. 12, where WS Set 2 is utilized, an even higher gain of 33.57% is recorded. These experiments validate that the initial pruning step reduces significantly the WS search space, hence resulting on the average in more improvement of the total execution time.

7. An optimal selection algorithm for a series of Web Services

Web Service Discovery up to now is designed for dealing with explicitly published changes to the registry data, which are typically done by designers. While these processes can be regarded as an approach to automatically handle changes in the registry, they do not represent a solution for the problem of dynamic service invocation or fault tolerance. A real world example of a Web Service selection and execution sequence is a travel plan management WS. In the case of travel plans, diverse Web Services need to be accessed one after the other, e.g., booking air travel, booking hotel room and so forth. Different paths for WSs' sequential consumption will exist as companies will compete by offering WSs with similar functionalities, e.g., different bus companies offering the same connection between two cities. An efficient way to manipulate and deliver Web Services' functionalities is therefore needed.

In this section we consider a generalized version of the Web Service selection problem. Before we proceed to the problem definition we provide some notation.

Let U, denote the set of all available WS. When the services must be executed in the strict order imposed by their subscripts, the set $W = \{w_1, w_2, \ldots, w_k\} \subseteq U$, is called a *Web Service execution sequence*. In other words w_i must be executed before w_j, for $1 \leqslant i < j \leqslant k$ and thus w_{i+1} is executed right after w_i. The Web Services constituting U are served from a set V, $|V| = n$ of servers, scattered across a wide area network. Each server $v \in V$,

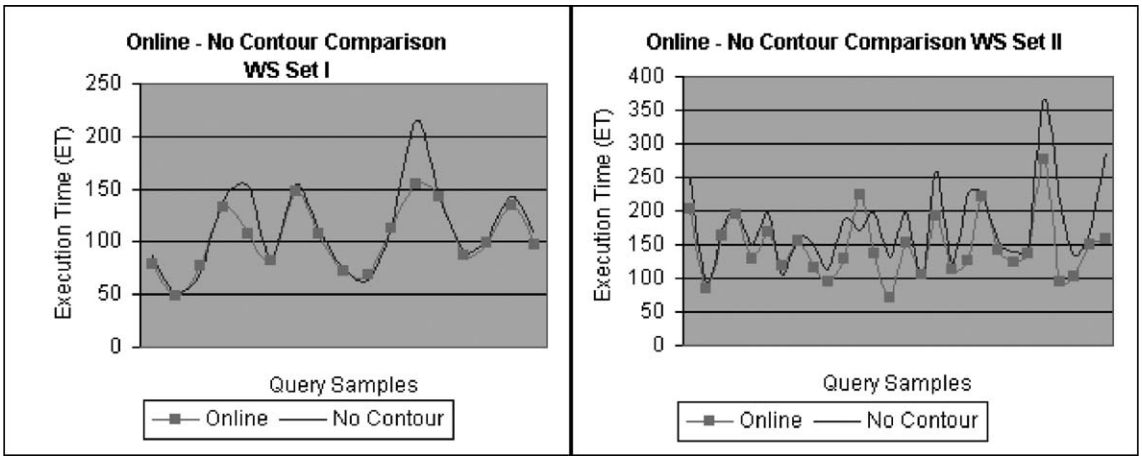

Fig. 12. Comparative evaluation of contour selection impact on WS Set 1 and 2.

hosts a set of Web Services $WS(v) \subseteq U$. Note that neither $WS(v) \subseteq W$ nor $WS(v) \subseteq W$, does necessarily hold. Along with each Web Service $ws \in WS(v)$ hosted in server $v \in V$, we have deduced an estimation of its execution time, $ET(ws, v)$, based on previously recorded execution schemes. The network latency $NL(v, w)$ between any two nodes v, w in V is also known. An *execution schedule* is a function ES: $V \times W \rightarrow \{0\} \cup \{1, 2, \ldots, k\}$, a function that defines which Web Service can be executed in what node and in what order (but can also output 0, if the service-node pair is incompatible).

Problem 1. Given a standard execution sequence W which does not change over time, a fixed access point s for W, a set of candidate servers V connected in a network along with the execution times $ET(ws, v)$, for each Web Service ws in $v \in V$, we want to compute an execution schedule ES for the sequence, where the execution schedule minimizes the total execution time for W.

We formulate the above problem as a *network optimization* problem. Hence, we consider the execution graph $G_W(\widetilde{V}, \widetilde{E})$ of W. The elements of $\widetilde{V}$ are produced as follows; for each pair (v_i, ws_j), v_i in set of network nodes V and ws_j in set of Web Services $WS(v_i)$ we construct a new virtual node $v_{i,j} \in \widetilde{V}$. In simple terms, $v_{i,j}$ denotes Web Service ws_j executed at node v_i. We choose G_W to be directed, thus an edge $e(v_{i,j}, v_{k,j+1})$ indicates that right after the execution of ws_j in v_i we choose node v_k to execute ws_{j+1}. Note that by defining edges in that way, we do not allow edges between nodes $v_{i,j}$ and $v_{k,j}$, i.e., nodes that serve the same Web Services. Furthermore, back edges, or in other words edges $e(v_{i,j}, v_{k,j-1})$ are not allowed, since in the execution sequence, ws_j is strictly followed by ws_{j+1}. Actually, once at a node $v_{i,j}$, the edges that leave $v_{i,j}$, correspond to all possible choices for the execution of the next Web Service, ws_{j+1}. A cost function $C: \widetilde{E} \rightarrow \mathscr{R}^+$ is also needed, to assign a non-negative weight to the edges of $\widetilde{E}$. For an edge $e(v_{i,j}, v_{k,j+1}) \in \widetilde{E}$, $C(e)$ corresponds to the execution time of ws_{j+1} in v_k plus the network latency to switch from v_i to node v_k, i.e.,

$$C(e) = ET(ws_{j+1}, v_k) + NL(v_i, v_k). \tag{2}$$

Note that since an edge $e \in \widetilde{E}$ may connect two virtual nodes that are distinct entities of the same real world computer, the latency term is zeroed out. The latter occurs in cases where two consecutive Web Services are executed on the same machine, so we do not need to switch between different machines.

We complement G_W with two more nodes, a source s (the access point) and a drain t. We add edges $(s, v_{i,1})$ for each possible $v_{i,1}$ and edges $(v_{i,k}, t)$ for all possible $v_{i,k}$ nodes. The edges emanating from s are assigned cost, equal to the cost of accessing $v_{i,1}$ from s. The edges leading to t are assigned zero cost. It is easy to compute the maximum possible size of G_W. $\widetilde{V}$ can contain at most $O(nk)$ nodes, in the case that each $v \in V$ hosts all the k services. In this worst case scenario, $\widetilde{E}$ will have $O(kn^2)$ edges. Inside a node $v_i \in V$ and from a node $v_{i,j}$ we can install at most one edge; that to $v_{i,j+1}$. We also need to produce the outgoing edges from $v_{i,j}$ to $v_{l,j+1}$ for all $v_{l,j+1} \in v_l$, where $l = 1, 2, \ldots, n$ excluding i. The number of nodes of this kind are $n - 1$. Therefore, from each $v_{i,j}$ we will have to produce at most $n - 1 + 1 = n$ edges and thus $O(kn^2)$, in total. Finally, from s and t we can install at most k edges. Hence, $|\widetilde{E}| = O(kn^2)$. Fig. 13 illustrates an example graph G_W.

Solution: With the above problem formulation, we have managed to represent a possible execution sequence as an s–t directed path. It follows that the optimal execution sequence, ES, is the *shortest* s–t path. Since the graph G_W is directed and acyclic, the minimum cost s–t path can be found in $O(|\widetilde{V}| + |\widetilde{E}|)$ time. This because the shortest path computation can be carried out

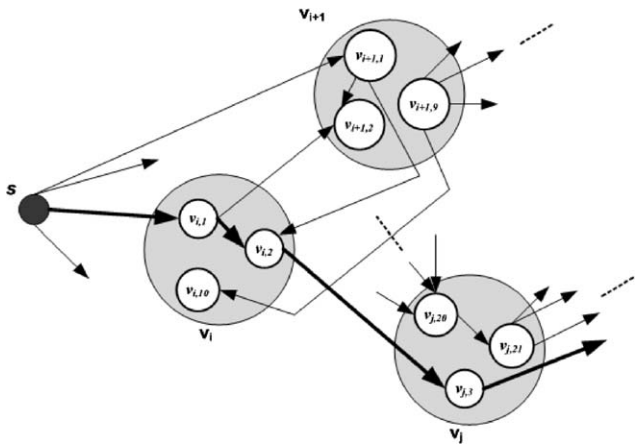

Fig. 13. An example graph G_W. Only partial information is shown. Nodes in gray are network nodes v_i which contains several virtual nodes $v_{i,j}$. Bold arrows indicate the shortest path.

in parallel with a topological sorting procedure in graph G_W (Cormen et al., 2001). Hence, in the worst case Web Service selection can be completed in $O(kn + kn^2) = O(kn^2)$ time.

Note that *ES* is valid globally through the network or in other words for all processes needing to execute W. This is true since the execution sequence is standard and all possible different access points for W will first have to access ws_1 and thus some $v_{i,1}$ node. Therefore, the construction of G_W need only be performed once for W. Nevertheless, the shortest s–t path may change if the access point changes. Therefore, a depth-first traversal of G_W is then necessary to recompute the new shortest path s–t path in G_W. Of course a different sequence W will require construction of another graph and subsequent computations.

8. Conclusions and future work

In this paper we were concerned with the problem of finding efficient solutions to the selection and binding to a WS or a series of Web Services, according to prespecified functional desiderata. We have introduced a novel and effective solution for ''online'' dynamic selection of Web Services. In particular, the introduced methods take into account quality of service (QoS) factors and maximize them in order to select the best matching Web Service at the moment of binding and consumption attempt. The mechanisms proposed have been designed to support the case of single Web Service selection. Furthermore, an extension has been also presented for cases of WS based workflows.

Apart from the theoretical support, we were able to justify our conjectures via extended experimental study performed using a telecommunication carrier processes, data and infrastructure as a test-bed. The results presented above clearly indicate that in most cases, our selection strategy is both more efficient and effective in comparison to the UDDI, in the sense that the total execution time resulting from the calculation of the optimal execution scheme is for the majority of practical cases, less than the time resulting from standard UDDI selection. Since the dynamic selection scheme proposed has proved to be efficient and applicable to large-scale business processes, we propose it as a useful approach for future web engineering enterprise solutions.

Future steps include different solutions about pruning first step selection beside the proposed contour selection. We also consider possible to achieve improved performance after integrating multicast based logic for collecting attribute values from the nodes. One more step forward is the adjustment of the proposed selection strategy for a peer-to-peer environment. Finally, Web Service selection algorithms may lead to further enhancements when dealing with recursive or flows of business Web Services.

References

Burstein, M.H., Hobbs, J.R., Lassila, O., Martin, D.L., McDermott, D.V., McIlraith, S.A., Narayanan, S., Paolucci, M., Payne, T.R., Sycara, K.P., 2002. DAML-S: Web Service description for the semantic web. In: Horrocks, I., Hendler, J.A. (Eds.), The Semantic Web—ISWC 2002, Proceedings of the First International Semantic Web Conference, Sardinia, Italy, June 9–12, 2002, Lecture Notes in Computer Science, vol. 2342. Springer, Berlin, pp. 348–363.

Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C., 2001. Introduction to Algorithms, second ed. The MIT Press/McGraw-Hill Book Company, Cambridge, MA/New York.

Deshpande, Y., Murugesan, S., Ginige, A., Hansen, S., Schwabe, D., Gaedke, M., White, B., 2002. Web engineering. J. Web Eng. 1 (1), 3–17.

Dong, X., Halevy, A.Y., Madhavan, J., Nemes, E., Zhang, J., 2004. Similarity search for Web Services. In: Nascimento, M.A., Özsu, M.T., Kossmann, D., Miller, R.J., Blakeley, J.A., Schiefer, K.B. (Eds.), Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31–September 3, 2004. pp. 372–383.

Garofalakis, J., Panagis, Y., Sakkopoulos, E., Tsakalidis, A., 2004. Web Service Discovery mechanisms: looking for a needle in a haystack? In: International Workshop on Web Engineering, Hypermedia Development and Web Engineering Principles and Techniques: Put them in use, in conjunction with ACM Hypertext 2004. Extended version submitted. Available from: <http://www.ht04.org/workshops/WebEngineering/HT04WE_Garofalakis.pdf>.

Li, Y., Zou, F., Wu, Z., Ma, F., 2004. Pwsd: A scalable Web Service Discovery architecture based on peer-to-peer overlay network. In: Yu, J.X., Lin, X., Lu, H., Zhang, Y. (Eds.), APWeb, Lecture Notes in Computer Science, vol. 3007. Springer, Berlin, pp. 291–300.

Liu, Y., Ngu, A.H.H., Zeng, L., 2004. Qos computation and policing in dynamic Web Service selection. In: Feldman, S.I., Uretsky, M., Najork, M., Wills, C.E. (Eds.), Proceedings of the 13th international conference on World Wide Web-Alternate Track Papers & Posters, WWW 2004, New York, NY, USA, May 17–20, 2004. pp. 66–73.

Makris, C., Panagis, Y., Sakkopoulos, E., Tsakalidis, A., 2004. Efficient search algorithm for large scale Web Service data. Tech. Rep. CTI TR 2004/09/01, RACTI.

Makris, C., Sakkopoulos, E., Sioutas, S., Triantafillou, P., Tsakalidis, A., Vassiliadis, B., 2005. Nippers: Network of interpolated peers for Web Service Discovery. In: 2005 IEEE International Conference on Information Technology: Coding and Computing (ITCC'05), vol. II, Las Vegas, Nevada. pp. 193–198.

Mehlhorn, K., 1984. Sorting and Searching Data Structures and Algorithms, vol. 1. Springer-Verlag, Berlin.

Microsoft, 2003. Uddi, windows 2003 implementation. <http://www.microsoft.com/windowsserver2003/technologies/webapp/uddi/default.mspx>.

Microsoft, 2004a. Perfomance counters and objects. <http://microsoft.com/windowsxp/home/using/productdoc/en/SAG_MPmonperf_06.asp>.

Microsoft, 2004b. Perfomance monitoring, browsing counters. <http://msdn.microsoft.com/library/en-us/perfmon/base/getting_counter_information.asp>.

Microsoft, 2005. C Sharp Programming Language Specification. <http://msdn.microsoft.com/library/en-us/csspec/html/CSharpSpecStart.asp>.

Moreau, L., Avila-Rosas, A., Miles, V.D.S., Liu, X., 2002. Agents for the grid: A comparison with Web Services (Part ii: Service discovery). In: Proceedings of Workshop on Challenges in Open Agent Systems. pp. 52–56.

Ouzzani, M., Bouguettaya, A., 2004. Efficient access to Web Services. IEEE Internet Comput. 8 (2), 34–44.

Overhage, S., Thomas, P., 2003. Ws-specification: Specifying Web Services using uddi improvements. In: Chaudhri, A.B., Jeckle, M., Rahm, E., Unland, R. (Eds.), Web, Web-Services, and Database Systems, NODe 2002 Web and Database-Related Workshops, Erfurt, Germany, October 7–10, 2002, Revised Papers, Lecture Notes in Computer Science, vol. 2593. Springer, Berlin, pp. 100–119.

Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K.P., 2002. Semantic matching of Web Services capabilities. In: Horrocks, I., Hendler, J.A. (Eds.), The Semantic Web—ISWC 2002, First International Semantic Web Conference, Sardinia, Italy, June 9–12, 2002, Proceedings, Lecture Notes in Computer Science, vol. 2342. Springer, Berlin, pp. 333–347.

Ran, S., 2003. A model for Web Services discovery with qos. SIGecom Exch. 4 (1), 1–10.

Rao, J., Su, X., 2004. A survey of automated Web Service composition methods. In: Cardoso, J., Sheth, A.P. (Eds.), Semantic Web Services and Web Process Composition, First International Workshop, SWSWPC 2004, San Diego, CA, USA, July 6, 2004, Revised Selected Papers. pp. 43–54.

Sajjanhar, A., Hou, J., Zhang, Y., 2004. Algorithm for Web Services matching. In: Advanced Web Technologies and Applications, Proceedings of the 6th Asia–Pacific Web Conference, APWeb 2004, Hangzhou, China, April 14–17, 2004. In: Yu, J.X., Lin, X., Lu, H., Zhang, Y. (Eds.), Lecture Notes in Computer Science, vol. 3007. Springer, Berlin, pp. 291–300.

Schlosser, M.T., Sintek, M., Decker, S., Nejdl, W., 2002. A scalable and ontology-based p2p infrastructure for semantic Web Services. In: Proceedings of the 2nd International Conference on Peer-to-Peer Computing (P2P 2002), 5–7 September 2002, Linköping, Sweden. IEEE Computer Society, Silver Spring, MD, pp. 104–111.

Schmidt, C., Parashar, M., 2004. A peer-to-peer approach to Web Service Discovery. World Wide Web 7 (2), 211–229.

Sivashanmugam, K., Verma, K., Sheth, A.P., Miller, J.A., 2003. Adding semantics to Web Services standards. In: Zhang, L.-J. (Ed.), Proceedings of the International Conference on Web Services, ICWS'03, June 23–26, 2003, Las Vegas, Nevada, USA. CSREA Press, pp. 395–401.

Stoica, I., Morris, R., Liben-Nowell, D., Karger, D.R., Kaashoek, M.F., Dabek, F., Balakrishnan, H., 2003. Chord: a scalable peer-to-peer lookup protocol for internet applications. IEEE/ACM Trans. Netw. 11 (1), 17–32.

Sycara, K.P., 2004. Dynamic discovery, invocation and composition of semantic Web Services. In: Vouros, G.A., Panayiotopoulos, T. (Eds.), Methods and Applications of Artificial Intelligence, Proceedings of the Third Helenic Conference on AI, SETN 2004, Samos, Greece, May 5–8, 2004, Lecture Notes in Computer Science, vol. 3025. Springer, Berlin, pp. 3–12.

UDDI, 2004. UDDI Version 3.0.2. <http://uddi.org/pubs/uddi-v3.0.2-20041019.htm>.

W3C, 2003. SOAP W3C Recommendation Documents. <http://www.w3.org/TR/SOAP>.

W3C, 2004a. Web Service Description Language. <http://www.w3.org/TR/wsdl>.

W3C, 2004b. Web Services Architecture. <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>.

Yu, T., Lin, K.-J., 2004a. The design of qos broker algorithms for qos-capable Web Services. Int. J. Web Service Res. 1 (4), 33–50.

Yu, T., Lin, K.-J., 2004b. Service selection algorithms for Web Services with end-to-end qos constraints. In: 2004 IEEE International Conference on Ecommerce Technology (CEC 2004), 6–9 July 2004, San Diego, CA, USA. pp. 129–136.

Yu, T., Lin, K.-J., 2005. A broker-based framework for qos-aware Web Service composition. In: 2005 IEEE International Conference on e-Technology, e- Commerce, and e-Services (EEE 2005), 29 March–1 April 2005, Hong Kong, China. pp. 22–29.

Zhang, J., Chung, J.-Y., Chang, C.K., Kim, S., 2004. Ws-net: A petri-net based specification model for Web Services. In: Proceedings of the IEEE International Conference on Web Services (ICWS'04), June 6–9, 2004, San Diego, California, USA. pp. 420–427.

Zhuge, H., 2002a. Clustering soft-devices in the semantic grid. IEEE Comput. Sci. Eng. 4 (6), 60–62.

Zhuge, H., 2002b. A knowledge grid model and platform for global knowledge sharing. Expert Syst. Appl. 22 (4), 313–320.

Zhuge, H., 2002c. Vega-kg: A way to the knowledge web. In: Proceedings of 11th International World Wide Web Conference (WWW2002). Available from: <http://www2002.org/CDROM/poster/53.pdf>.

Zhuge, H., Liu, J., 2004. Flexible retrieval of Web Services. J. Syst. Softw. 70 (1–2), 107–116.

Christos Makris was born in Greece, in 1971. He graduated from the Department of Computer Engineering and Informatics, School of Engineering, University of Patras, in December 1993. He received his Ph.D. degree from the Department of Computer Engineering and Informatics, in 1997. He is now an Assistant Professor in the same Department. His research interests include Data Structures, Web Algorithmics, Computational Geometry, Data Bases and Information Retrieval. He has published over 40 papers in various scientific journals and refereed conferences.

Yannis Panagis was born in Greece, in 1978. He is currently a PhD candidate at the Computer Engineering and Informatics Department at the University of Patras and a member of the Research Unit 5 of the RA Computer Technology Institute. Yannis holds an M.Sc. from the same Department, where he has also completed his undergraduate studies. His interests span the areas of Data Structures, String Processing Algorithms and Web Engineering, where he has published papers in international journals and conferences. He has also co-authored two book chapters.

Evangelos Sakkopoulos was born in Greece, in 1977. He is currently a PhD candidate at the Computer Engineering and Informatics Department, University of Patras, Greece and a member of the Research Unit 5 of the RA Computer Technology Institute. He has received the M.Sc. degree with honors and the diploma of Computer Engineering and Informatics at the same institution. His research interests include Web Services, Web Engineering, Web Usage Mining,

Web based Education, Web Services, Web Searching and Intranets. He has more than 20 publications in international journals and conferences at these areas.

Athanasios K. Tsakalidis, Computer-Scientist, Professor of the University of Patras. Born 27.6.1950 in Katerini, Greece. Studies: Diploma of Mathematics, University of Thessaloniki in 1973. Diploma of Informatics in 1980 and Ph.D. in Informatics in 1983, University of Saarland, Germany. Career: 1983–1989, researcher in the University of Saarland. He has been student and cooperator (12 years) of Prof. Kurt Mehlhorn (Director of Max-Planck Institute of Informatics in Germany). 1989–1993, Associate Professor and since 1993 Professor in the Department of Computer Engineering and Informatics of the University of Patras. 1993–1997 and 2001–today, Chairman of the same Department. 1993–today, Member of the Board of Directors of the Research Academic Computer Technology Institute (RACTI), 1997–today, Coordinator of Research and Development of RACTI, 2004–today, Vice-Director of RACTI. He is one of the contributors to the writing of the "Handbook of Theoretical Computer Science" (Elsevier and MIT-Press 1990). He has published many scientific articles, having an especial contribution to the solution of elementary problems in the area of data structures. Scientific interests: Data Structures, Computational Geometry, Information Retrieval, Computer Graphics, Data Bases, and Bio-Informatics.