2009 International Conference on Web Information Systems and Mining

# WSPRC: An Adaptive Model for Query Optimization over Web Services

Dunlu Peng, Chen Li, Feng Zhu, Ning Zhang
School of Optical-Electrical and Computer Engineering,
University of Shanghai for Science and Technology,
Shanghai 200093, China
{dlpeng@usst.edu.cn, lcroy@163.com, percyzu@126.com, zn200xue2@163.com}

***Abstract*—Query optimization over Web services is very important for data integration with Web services and has gained much attention in recent year. In this work, we propose an adaptive query optimization model for Web services, that is, Web Service Profiler-Reoptimizer-Cache (WSPRC). One of the core components of the model is Reoptimizer which is based on the adaptive greedy algorithm (A-Greedy) and analyzes the implementation of Web service query optimization process. We compare the traditional Greedy with A-Greedy from several aspects. The comparision and experimental results show that the WSPRC model and A-Greedy improve the efficiency of querying optimization over Web services.**

***Keywords: Web service; query optimization; Adaptive; WSPRC; A-Greedy***

## I. INTRODUCTION

Web services query optimization has gained much attention in recent years. Non-adaptive greedy algorithm can improve the performance of Web services and enhance the efficiency of service implementation. However, the traditional greedy algorithm for Web services can not meet the needs of the query optimization for the distribution of data behind Web services and the emergence of data stream.

With the emergence of Adaptive Query Processing (AQP) [1], systems based on AQP are widely applied. The nature of the adaptive query is to adjust the implementation of query process dynamically according to the query feedback. Web services may have to optimize across multiple queries or need to adaptively implement a query. These requirements are the properties of adaptive query processing. Using adaptive technology, query optimization over web services will affect the query plan being implemented or the scheduled operations on the local DBMS. In this way, it improves query performance and effectively handles the distribution and the data stream.

The rest of this paper is organized as follows. Section 2 gives the basic concept of AQP, and in Section 3, we propose a model for adaptive query optimization over web services, that is, Web Service Profiler-Reoptimizer-Cache (WSPRC). Based on the introduction of Adaptive Greedy Algorithm[1][2] (A-Greedy), in Section 5, we combine the A-Greedy algorithm with query optimization over Web services and explain the process of query optimization. We evaluate the performance of query optimization process over Web services using our approach in Section 6. Finally, we draw our conclusions in Section 7.

## II. ADAPTIVE QUERY PROCESSING

We will first give the definition of AQP, and then introduce the relationship between AQP and Web service.

### *A. AQP Definition*

AQP is the process of query optimization which adjusts the implementation of query plan dynamically in accordance with the query feedback.

### *B. AQP and Query Optimization over Web Services*

When customers need to invoke Web services[3], they are usually referred to the order in which they query the Web services and other issues (for example, response time). In order to improve the performance of query over Web services, how to optimize query process effectively over Web services has become a key issue. AQP technology can not only be used to process the data stream and reduce the complexity of the query process, but also to improve the query performance.

## III. WSPRC

In this section, the WSPRC Model, which uses AQP technology to improve query optimization over Web services, and its specific process are described.

### *A. WSPRC Model*

The main idea underlying the Web Service Profiler-Reoptimizer-Cache model is that when users try to find Web services through the model, it will execute a query for retrieving Web services and analyze them. Meanwhile, the model optimizes the query process by developing a query plan and saving it in the cache. In such way, Web services will be invoked in accordance with the query plan. Figure 1 describes the WSPRC model.

### *B. WSPRC Model Processing*

We explain WSPRC model processing with an example.

*Example 1:* Assumes that an enterprise needs to release medical allowances to employees' Social Security whose working age is greater than 10 years.

The following are four related Web services.

$WS_1$: Employee ID → Identity Card



978-0-7695-3817-4/09 $26.00 © 2009 IEEE
DOI 10.1109/WISM.2009.78

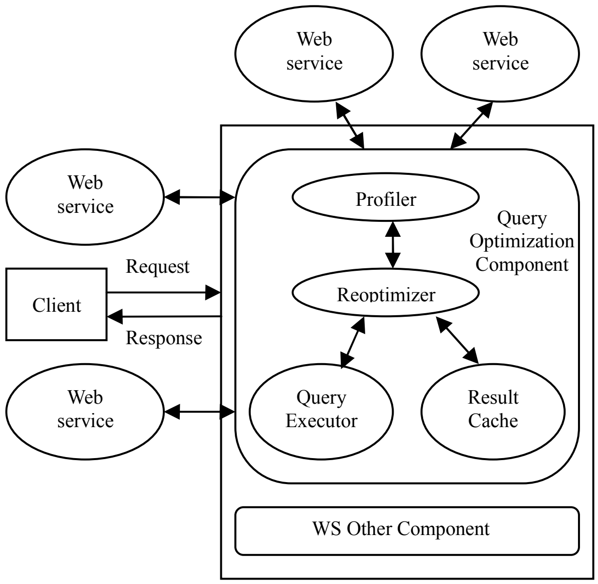



Figure 1. WSPRC Model

$WS_2$: Identity Card → Working Age

$WS_3$: Identity Card → Social Security Number

$WS_4$: Social Security Number → Credit Card Number

There are two query plans in this example, that is, Linear and Parallel. We illustrate the two plans in Figure 2.

The following pseudo-code describes the query process with WSPRC model. Then notations are depicted as below:

- $WS_1, WS_2, \ldots, WS_i, \ldots, WS_n$ ($1 \leq i \leq n$)
- $A_j$ : projected attributes ($1 \leq j \leq n$)
- $P_j(A_j)$ : predicate applied on attribute
- *ITQ* : input tuple queue
- **A-Greedy** : Adaptive Greedy Algorithm
- *RC* : Result Cache

```
While (ITQ! = null)
  evaluate RC; //evaluate WS query plan from RC
    if (find plan)//find the plan from RC
      then Execute the Query Plan of RC;
    else//plan not found
      While (use A-Greedy to find WSi)
        invoke WSi;
          for each return tuple tj;
            apply all predicates Pj(Aj) on tj;
            if tj satisfies all predicates
              write tj UNION output
      Execute the query plan
    save the plan and the results in RC
```

*C. Analysis of WSPRC Model*

Now, we analyze its key components. Detail of A-Greedy Algorithm will be discussed in Section 5.

*1) Result Cache:* Adding Result Cache component to the WSPRC model to cache query results for the Web services which have been invoked before. In this way, it not only saves the querying time of Web services, but also improves the efficiency in the implementation of WS.

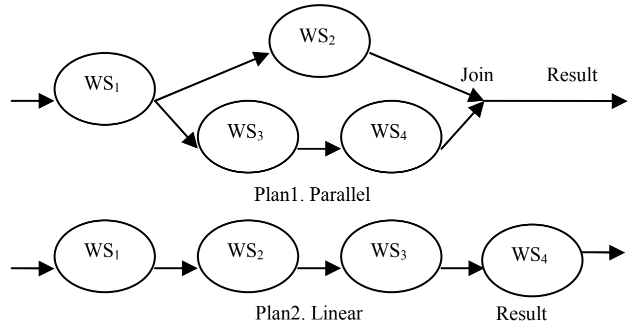



Figure 2. WS Query Plan (Parallel & Linear)

*2) Profiler:* In some Web services systems (WSMS[4]), Profiler exists and its aim is to analyze the response time of Web services and related statistical data. In this work, the Profiler of WSPRC model taking the two previous common roles will continue to search and analyze the Web services which have been registered over the Internet.

*3) Reoptimizer:* Reoptimizer is the core component of the WSPRC model. It uses the A-Greedy to analyze the Web service related information received from the Profiler.

Three components aforementioned are the core parts of the WSPRC model. The model improves the efficiency and reduces the query time of the Web services to some extent.

## IV. IMPACT FACTORS ON WEB SERVICES QUERY

In this section, we consider three key factors that influence the query performance.

*A. Precedence Constraints*

In any feasible execution plan for the query, if $WS_i$ must precede $WS_j$, then there is a precedence constraint $WS_i \rightarrow WS_j$[4-6].

*B. Filter*

In the reference [1], selectivity is used to indicate that when $WS_i$ receives a tuple, it will apply relevant predicates on attributes to get results. This work uses the filter to measure the $WS_i$ capacity of filtering input tuples.

*C. Tuple Processing Time*

The tuple processing[4] is starting at time $t_1$ when the WS receives a tuple, and ending at time $t_2$ when the results are returned, so tuple processing time is the time difference between $t_2$ and $t_1$.

## V. A-GREEDY FOR QUERY OPTIMIZATION

Based on the introduction of Pipelined Filters problem and Adaptive Loop framework[1] which A-Greedy should follow, we provide the main ideas of A-Greedy and query optimization process over Web services.

*A. Pipelined Filters Problem*

Currently, there are a lot of researches on Pipelined Filters [7,8]. The so-called Pipelined Filters is using a series of filters to deal with continuous input tuples flow.

*B. Adaptive Loop*

AQP technology based on Adaptive Loop framework includes four components: measurement, analysis, planning and actuation. A-Greedy follows this framework to invoke the Web services, and optimizes the query process.

*C. Main Ideas of A-Greedy*

Before we discuss the main ideas of A-Greedy, the relevant definitions are given as follow:

$F_i$: Filter 1, 2, $i$,…, $n$ ($1 \le i \le n$)

*CP (i,j)*: Conditional Probability, that is, $F_i$ will drop a tuple $t$ which was not dropped by any of $F_1, F_2, \dots, F_j$ ($1 \le i \le n, j = i - 1$).

$CT_i$: Cost Time, that is, tuple processing time ($1 \le i \le n$)

$O$: Total Cost[2], the formula is as follow.

$$O = \begin{cases} \sum_{i=1}^{n} \left( CT_i \prod_{j=1}^{i-1} \left(1 - CP(j, j-1)\right) \right) & i > 1 \\ \sum_{i=1}^{n} CT_i & i = 1 \end{cases} \tag{1}$$

*GI*: Greedy Invariant [2], A-Greedy maintains an order that satisfies the *GI*. By weighing the conditional probability and tuple processing time to compare the Web services query cost, less cost Web service will be given a priority to be invoked. Specific formula is as follows:

$$\frac{CP(i,i-1)}{CT_i} \ge \frac{CP(j,i-1)}{CT_j}, 1 \le i \le j \le n \tag{2}$$

*GI* formula does not suit for the following situations:

- There are precedence constraints among WS.
- There are no precedence constraints and no relevant predicates, that is, parallel plan.

The main ideas of A-Greedy is continuous monitoring of the current query cost (O) of Web services, according to the order from cheap to expensive to develop the best pipelined query plan for Web services, and adjust query plan dynamically according to the feedback information of Web services and changes of current input tuples.

*D. Optimization Process of A-Greedy*

The implementation of A-Greedy process can be divided into two phases: the construction of the initial query plan and the plan optimization. The following will introduce these two phases.

*1) Constructing the initial Web services query plan.* Such as *Example 1,* in accordance with the sequence of invokes, we construct the initial query plan, usually it is more like a parallel plan (see Figure 2 plan1).,and we also modify *Example 1* by adding another Web service, that is, $WS_5$, to find the name of the area where qualified employee's social security belongs to (see Figure 3).

$WS_5$: Social Security Number → District Name

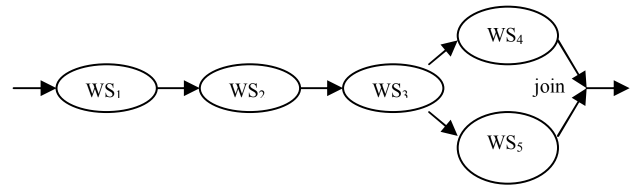



Figure 3. WS Query Plan added $WS_5$

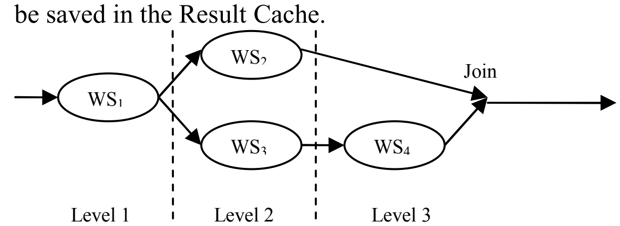
*2) Classification of Web services.* We use the (1) and (2) for evaluation and analysis of the Web services query cost. Levels are as follows as shown in Figure 4.

*a) Root-level and precedence constraints.* As there is a specified relationship between these services, the A-Greedy would ignore the root-level (Level 1) and precedence constraints optimization.

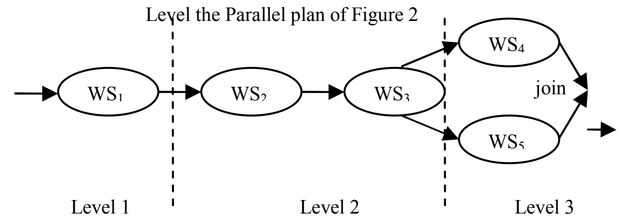
*b) Without precedence constraints but with relevant predicates.* In the first query plan of Figure 4, $WS_2$ and $WS_3$ are belong to level 2, and there are no precedence constraints between them but contains a predicate which means working age is greater than 10 years in $WS_2$. So if we call $WS_2$ before $WS_3$ that will reduce the amount of data which $WS_3$ should deal with. A-Greedy would adjust the location of $WS_2$ and $WS_3$. This is shown in plan 2 of Figure 2.

*c) Without precedence constraints and without relevant predicates.* In Figure 4, level 3 in the second query plan, there are no precedence constraints and no relevant predicates between $WS_4$ and $WS_5$, that is, the results coming from $WS_4$ and $WS_5$ are independent of each other. At this point, A-Greedy would be used to take parallel plan to invoke $WS_4$ and $WS_5$; this will improve the concurrency of Web services and reduce the implementation time and improve efficiency

*3) Execute step 2 repeatedly until the required Web services have been optimized in a certain order.* Then the query plan will be executed and the results will be sent to user. At the same time, the query plan and results will also be saved in the Result Cache.




Level the Parallel plan of Figure 2




Level the query plan of Figure 3

Figure 4. Level the Query Plan

After that, since Web services and the data information are always under dynamic changes, the original query optimization plan may no longer be appropriate, thus bringing out a need for re-optimization. The specific re-optimization process is as follows:

*4) Re-optimization caused by input information change.* The measurement component of Adaptive Loop framework will evaluate input tuples information in a period of time, execute the query plan saved in Result Cache earlier and then analyze the results. If the query cost increases, it would adjust Web services query plan, and then go to step 6.

*5) Re-optimization caused by Web services change.* During query Web services, measurement component of Adaptive Loop framework will evaluate Web services which should be invoked. If the Web service changes on its own, and leads to cost increasing (such as the Web service response time went longer) or reducing that need to adjust the location of Web services, and it then goes to step 6.

*6) Execute re-optimization Web Services query plan and return the results to user and save the results and the optimized query plan in Result Cache.* If there is still data stream coming then it goes to Step 4 for evaluation, and analysis, otherwise the query process of Web service will end.

The above is the A-Greedy optimization process. A-Greedy takes Adaptive Loop framework, and provides real-time measurement and analysis and develops a new query plan, which is impossible with the original Greedy.

## VI. EXPERIMENTS

This section evaluates the performance of original Greedy and A-Greedy on query Web services based on the following three aspects.

- Construct the initial Web service query plan
- Input tuples information change
- Web services change

*Example 2:* Suppose an enterprise needs to release medical allowances to employees' credit card which are associated with Social Security, and these employees' working age are greater than 10 years and they have medical insurance. Likewise, the name of the area where qualified employee social security belongs to should be found.

The following are six related Web services.

$WS_1$: Employee ID $\rightarrow$ Identity Card
$WS_2$: Identity Card $\rightarrow$ Working Age
$WS_3$: Identity Card $\rightarrow$ Medical Insurance
$WS_4$: Identity Card $\rightarrow$ Social Security Number
$WS_5$: Social Security Number $\rightarrow$ Credit Card Number
$WS_6$: Social Security Number $\rightarrow$ District Name

Table 1 is tuple processing time (CT) of $WS_1$ ~ $WS_6$. Table 2 is the selectivity of $WS_1$ ~ $WS_6$, that is, 1 – CP.

TABLE I. TUPLE PROCESSING TIME (CT)

| Web Service | $WS_1$ | $WS_2$ | $WS_3$ | $WS_4$ | $WS_5$ | $WS_6$ |
|---|---|---|---|---|---|---|
| **Cost Time** | 5.5 | 4.1 | 3.3 | 3.5 | 5.4 | 7.6 |

TABLE II. SELECTIVITY (1-CP)

| Web Service | $WS_1$ | $WS_2$ | $WS_3$ | $WS_4$ | $WS_5$ | $WS_6$ |
|---|---|---|---|---|---|---|
| **Selectivity** | 1 | 0.63 | 0.27 | 0.71 | 0.86 | 0.91 |

### *A. Construct the initial WS query optimization plan*

*1)* Find the precedence constraints among Web services. The analysis of *Example 2* is as follows:

*a)* With precedence constraints: $WS_1 \rightarrow WS_2$, $WS_1 \rightarrow WS_3$, $WS_1 \rightarrow WS_4$, $WS_4 \rightarrow WS_5$, $WS_4 \rightarrow WS_6$.

*b)* Parallel Web services: ($WS_2$, $WS_3$, $WS_4$) and ($WS_5$, $WS_6$) can be paralleled.

*2)* Construct the initial Web service query plan, and level them. Figure 5 is shown.

*3)* According to the (1) and (2), we calculated the cost of $WS_1$ ~ $WS_6$ query. The following is the calculation ("I" means input tuples):

*a)* Level 1 query cost. Cost ($WS_1$) = 5.5 * 1 * I = 5.5I

*b)* Level 2 query cost. There are no precedence constraints among $WS_2$, $WS_3$, and $WS_4$, but they have predicates on the results which $WS_1$ has returned. So Greedy and A-Greedy will use (2) on $WS_2$, $WS_3$ and $WS_4$ (see table 3), and then use (1) to calculate the query cost.

According to table 3, the possible order of $WS_2$, $WS_3$ and $WS_4$ is $WS_3 \rightarrow WS_2 \rightarrow WS_4$. Then we calculate the cost for query path.

Initial Query plan: Cost ($WS_2$, $WS_3$, $WS_4$) = 11.3I
Linear plan: Cost ($WS_3 \rightarrow WS_2 \rightarrow WS_4$) = 5.00235I

From table 3 and the cost calculation we know that the best query plan in level 2 is $WS_3 \rightarrow WS_2 \rightarrow WS_4$, and the minimum cost is 5.00235I.

*c)* Level 3 query cost. There are no precedence constraints or relevant predicate between $WS_5$ and $WS_6$. So they can be paralleled and needn't change the access path. $WS_5$ and $WS_6$ are in the last level 3, so they always are invoked at the end of the query plan and that makes their cost be a constant.

*4)* Construct the optimized query plan shown in Figure 6.

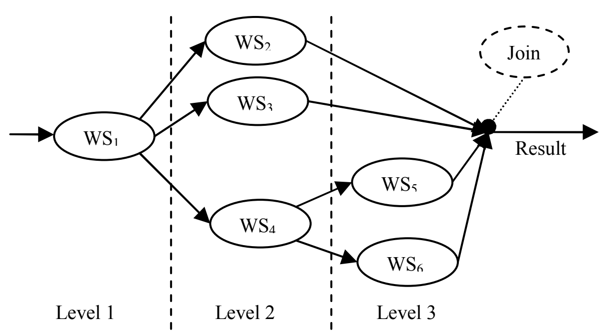



Figure 5. Initial Web Service Query Plan

TABLE III. APPLY GREEDY INVARIANT ON LEVEL 2

| Web Service | $WS_2$ | $WS_3$ | $WS_4$ |
|---|---|---|---|
| **CP(i,i-1)/CTi** | 0.09024 | 0.22121 | 0.08285 |

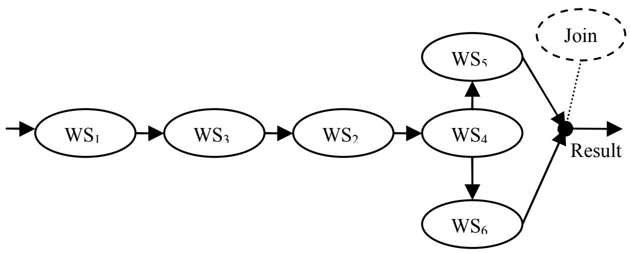



Figure 6. Optimized Query Plan

It is a similar optimization process that using A-Greedy or Greedy to construct the initial query plan.

Cost (WS) = Cost ($WS_1$) + 5.00235I + Cost ($WS_5$, $WS_6$)

### B. *Input tuples information change*

After we find the optimized query plan (see Figure 6), with increasing employees' ID that 90% employees having medical insurance, that is, $WS_3$ filter CP = 0.1.

*1)* Greedy is not adaptive, so it will continue to use the old optimized query plan and cost is as follows:

Cost $(WS)_{Greedy}$ = Cost ($WS_1$) + 8.9745I + Cost ($WS_5$, $WS_6$)

*2)* A-Greedy will evaluate $WS_2$, $WS_3$ and $WS_4$ cost again when it detects the changes of input tuples. Since the cost of $WS_1$, $WS_5$ and $WS_6$ does not change, we can still consider them as constants. So we only have to re-evaluate the cost of Level 2:

According to table 4, the possible order of $WS_2$, $WS_3$ and $WS_4$ is $WS_2 \rightarrow WS_4 \rightarrow WS_3$. Then we calculate the cost for query path.

Cost ($WS_2 \rightarrow WS_4 \rightarrow WS_3$) = 7.7810I

From table 4 and the cost calculation we know, using A-Greedy, the total cost of query plan is as follows:

Cost $(WS)_{A\text{-}Greedy}$ = Cost ($WS_1$) + 7.7810I + Cost ($WS_5$, $WS_6$)

### C. *Web services change*

We still modify *Example 2*. After we find the optimized query plan (see Figure 6), for some unknown reasons, $WS_2$ needs more time to process a tuple, that is, increasing tuples processing time (CT = 6).

*1)* Greedy is not adaptive, so it will continue to use the old optimized query plan and cost is as follows:

Cost $(WS)_{Greedy}$ = Cost ($WS_1$) + 5.51535I + Cost ($WS_5$, $WS_6$)

*2)* A-Greedy finds that $WS_2$ needs more time to process a tuple. So it will evaluate $WS_2$, $WS_3$ and $WS_4$ cost again. We still consider the cost of $WS_1$, $WS_5$ and $WS_6$ as constants. Now we re-evaluate the cost of Level 2:

TABLE IV. APPLY GREEDY INVARIANT ON LEVEL 2

| **Web Service** | ***$WS_2$*** | ***$WS_3$*** | ***$WS_4$*** |
|---|---|---|---|
| **$CP(i,i-1)/CT_i$** | 0.09024 | 0.030303 | 0.08285 |

TABLE V. APPLY GREEDY INVARIANT ON LEVEL 2

| **Web Service** | ***$WS_2$*** | ***$WS_3$*** | ***$WS_4$*** |
|---|---|---|---|
| **$CP(i,i-1)/CT_i$** | 0.061666 | 0. 22121 | 0.08285 |

According to table 5, the possible order of $WS_2$, $WS_3$ and $WS_4$ is $WS_3 \rightarrow WS_4 \rightarrow WS_2$. Then we calculate the cost for all query paths.

Cost ($WS_3 \rightarrow WS_4 \rightarrow WS_2$) = 5.3952I

From table 5 and the cost calculation we know that the less cost path in level 2 is $WS_3 \rightarrow WS_4 \rightarrow WS_2$, so using A-Greedy, the total cost of query plan is as follows:

Cost $(WS)_{A\text{-}Greedy}$ = Cost ($WS_1$) + 5.3952I + Cost ($WS_5$, $WS_6$)

From the comparisons of the three aspects above, we know that the A-Greedy performance on query Web services is better than the Greedy, and it can be better adapted to Web services and data stream changes.

## VII. CONCLUSIONS

This paper provides a Web Service Profiler-Reoptimizer-Cache (WSPRC) model for adaptive query optimization over web services and analyzes its three components. We focus on the core component Reoptimizer based on pipelined A-Greedy algorithm. From with precedence constraints or not and relevant predicates, it finds the best query plan for Web services. Experimental results shows that after introducing AQP technology, A-Greedy is better than Greedy for adapting the changes of Web services, evaluating the current Web services query plan real-time and developing a better query plan.

## ACKNOWLEDGEMENT

We would like to express our special thanks to Mingzhu Jin and Hui Guo for their helpful suggestions to improve the paper. This work was supported by the Innovation Program of Shanghai Education Commission under Grant No. 08YZ98.

## REFERENCES

[1] A. Deshpande, Z. Ives and V. Raman. Adaptive Query Processing [J]. Foundations and Trends in Databases, 2007,1(1):1-140.

[2] S. Babu, R. Motwani, K. Munagala, I. Nishizawa, and J. Widom, "Adaptive ordering of pipelined stream filters," in SIGMOD'04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data,(New York, NY, USA), pp. 407–418, ACM Press, 2004.

[3] Web Services, 2002. http://www.w3c.org/2002/ws.

[4] U. Srivastava, K. Munagala, J. Widom, and R. Motwani, "Query optimization over web services", in VLDB'06: Proceedings of the 32nd international conference on Very large data bases, pp. 355–366, VLDB Endowment, 2006.

[5] Burge J, Munagala K, Srivastava U. Ordering Pipelined Operators with Precedence Constraints [EB/OL]. (2009-05-15). http://infolab.stanford.edu/~usriv/papers/precconst.pdf

[6] Xu Shu Hua,Jiang Wen,Huang Zhi Gang. Algorithm of Web Services query based on bottleneck cost [J]. Computer Applications, 2007,27(8):1997-2000.

[7] K. Munagala, S. Babu, R. Motwani, and J. Widom, "The pipelined set cover problem," in ICDT '05: Proceedings of the 10th International Conference, Edinburgh, UK, pp. 83–98, 2005.

[8] A. Condon, A. Deshpande, L. Hellerstein, and N. Wu, "Flow algorithms for two pipelined filter ordering problems," in PODS '06: Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, (New York, NY, USA), pp. 193–202, ACM Press, 2006.