Mobile Information Systems 1 (2005) 167–184
IOS Press

Mobile join operators for restricted sources[1]

Belgin Ozakar[a], Franck Morvan[b] and Abdelkader Hameurlain[b]
[a]*Izmir Institute of Technology, Urla, Izmir, Turkey*
Tel.: +90 232 750 65 32; Fax: +90 232 750 65 05; E-mail: belginozakar@iyte.edu.tr
[b]*Institut de Recherche en Informatique de Toulouse IRIT, Université Paul Sabatier, 118, Route de Narbonne, 31062 Toulouse Cedex, France*
Tel.: +33 (0) 5 61 55 82 48 6325; Fax: +33 (0) 5 61 55 62 58; E-mail: {morvan, hameur}@irit.fr

Abstract. We consider the problem of query execution when there is limited access to the relations, i.e. when binding patterns require values to be specified in order to get data from the relation. This problem is common in virtual data integration systems where there are heterogeneous sources with various restricted access patterns and query capabilities. Another problem is the lack of the statistical information about the sources and occurrence of unpredictable events. We introduce two mobile join operators, MDJoin and SMDJoin which are designed for restricted sources and implemented using 'mobile agents' in order to benefit from their autonomous and reactive characteristics. Mobile operators of restricted sources are capable to deal with restricted sources and react to the variations between the compile-time estimations and run-time computations of data during query execution. The difference between the two new query operators lies in their level of adaptation ability to the execution environment. Performance results show that mobile agent-based approach at operator level can lead to a significant reduction in response time with restricted sources.

Keywords: Virtual data integration, restricted sources, adaptive query execution, mobile agents, mobile dependent join operators, performance evaluation

1. Introduction

The goal of virtual data integration systems is to support seamless access to heterogeneous data sources. They provide uniform interface to users freeing them from having to locate the sources relevant to their queries, to interact with each source in isolation and to combine the data from the different sources. In heterogeneous environment, just as World Wide Web, data sources have diverse access restrictions and query capabilities. Suppose we want to buy a book; we fill a form giving the ISBN or title of the book in order to see the stock-on-hand or the price values related to the book. Due to the reasons of privacy or performance issues, we cannot retrieve information from these sources without providing input values. Dealing with such restricted sources, brings out various issues to be studied in terms of providing simple concept of modeling for the source descriptions and query capabilities, query execution plan generation solving the complexity caused by the source restrictions and query execution operators that are able to work efficiently despite of the source restrictions.

In virtual data integration in large scale as well, query optimization is based on cost estimation depending on the statistical information coming from the sources [3]. However cost estimation is

[1]This work was supported in part by the French ministry for research "ACI Masses de Données 2004", GGM project, grant number 04 2 32.

difficult in such an environment where (i) data sources do not export needed statistical information, (ii) the statistical information is not up-to-date, or (iii) unpredictable events occur during query execution. So query optimizer would need adaptive mechanisms to avoid missing optimal query execution with missing or obsolete statistics and in the occurrence of any unforeseen event.

Related work on virtual data integration in large scale can be classified with regard to its primary focus as being on restricted sources or adaptive query processing. The related work whose primary focus is restricted sources consists of three groups of efforts; (i) modeling the heterogeneity of the sources in terms of access restrictions and query capabilities [18,25], (ii) generating valid, efficient and complete query execution plans in the level of the optimizer [4,13,15,19,21,25] and (iii) developing query execution operators handling restricted access [5,17]. The related work with the primary focus as adaptive query processing can be grouped as centralized [2,7,8,11,12,28] or decentralized [1,9,10,14, 16,20,23,24,26] approaches. To our knowledge there is no work to demonstrate the impact of *software mobility* in decentralized adaptive query processing at operator level concerning the source restrictions in a virtual data integration environment. The adaptation of our operators to mobile device environment [9, 14,16,23,24] is outside the scope of this paper.

The autonomy and reactivity provided by the mobile agent model and use of mobile agents as a tool for query execution provide us the following advantages: (i) execute join operations as close as possible to data, thus limit the volume of transferred data, (ii) decentralize the control of dynamic optimization, so that network bottleneck created by centralized monitoring is avoided, (iii) react to the variations between the compile-time estimations and run-time computations of data.

Our challenge is to contribute to the third group of efforts focusing on restricted sources and decentralized adaptive query processing. We by develop efficient query execution operators which are capable to work with restricted sources and which are, at the same time, adaptive to the run time parameters of data (i.e. size, cost of sending, computing and receiving). Autonomous and reactive characteristics of mobile agents gave us the inspiration to develop self-adaptive algorithms for join operators with the advantage of decentralizing the monitoring. We assume that heterogeneity of the sources can be abstracted by means of a virtual data integration system based on mediators and wrappers as proposed by [30]. In modeling these sources, we depend on the concept 'binding patterns' as explained in [17].

Hence we want to develop a mobile execution model that is suitable for restricted sources. Our approach is to:

– demonstrate basic operator needs of restricted sources with an example,
– embed the strength of mobile execution plan into the join operator of restricted sources and create two new mobile join operators: MDJoin and SMDJoin,
– evaluate the performance of mobile join operators designed for restricted sources with respect to (i) variation between the compile-time estimation and run-time computation of the sizes of data, (ii) CPU frequency and (iii) communication bandwidth.

The rest of the paper is organized as follows; Section 2 introduces two basic operators of restricted sources, DAccess and DJoin. In Section 3 the algorithms of mobile dependent joins, MDJoin and SMDJoin are explained. Section 4 demonstrates the performance evaluation results; exploring the comparison of the behaviors of the DJoin, MDJoin and SMDJoin operators depending on different sizes, CPU frequencies and communication bandwidths. Section 5 is a survey of the related work on restricted sources and adaptive query processing. Section 6 is discussion of the performance evaluation results and Section 7 is our conclusion and remarks for the future work.

Telephone

name	***telNo***
Ahmet Hoşgör	90-232-8990786
Mehmet Yılmaz	90-312-7645673
Hüseyin Kaçar	90-212-8978990
Leyla Koruk u	90-232-7506530

Address

telNo	***Address***
90-232-8990786	234 sok. 31 Bostanlı IZMIR
90-312-7645673	556 sok. 45 ˙ankay a ANKARA
90-212-8978990	789 sok. 43 Etiler ISTANBUL
90-232-7506530	786 sok. 1 IZMIR

Fig. 1. Sample database.

2. Operators for restricted source access

Suppose we have the following example as seen in Fig. 1; there are two tables holding data related to the telephone subscribers; *Telephone*(*name, telNo*), *Address*(*telNo, address*). We assume that this database is distributed which means Telephone table is located on a site different than the location of the Address table but the tables are not fragmented. We also assume that those two tables are modeled with the concept of ‘binding patterns’ as introduced in [25].

Binding patterns can be attached to the relational table to describe its access restrictions due to the reasons of confidentiality or performance issues. A binding pattern for a table $R(X_1, X_2, \ldots, X_n)$ is a partial mapping from $\{X_1, X_2, \ldots, X_n\}$ to the alphabet $\{b, f\}$ [17]. For those attributes mapped to ‘b’, the values should be supplied in order to get information from R while the attributes mapping to ‘f’ do not require any input in order to return tuples from R. If all the attributes of R are mapped to ‘f’ then it is possible to get all the tuples of R without any restriction (e.g. with a relational scan operator).

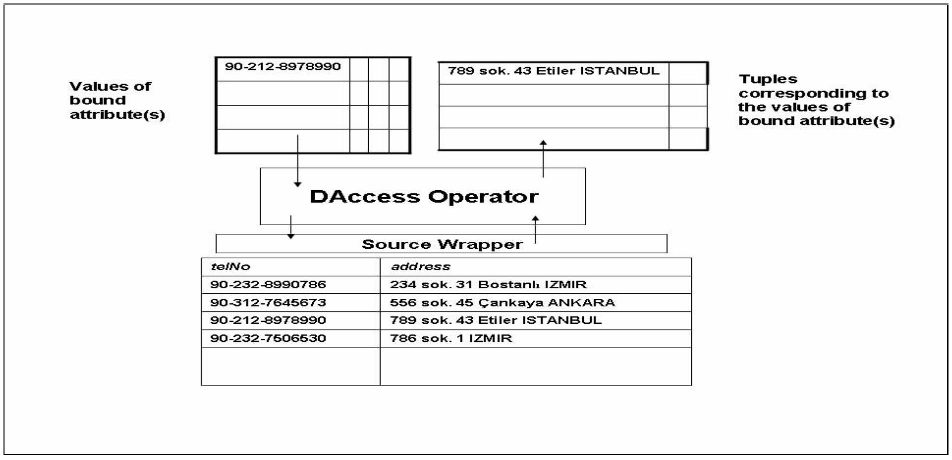
If we know that the binding patterns of the tables of our sample database are as follows *Telephone*(*name*^{f} *telNo*^{f}), *Address*(*telNo*^{b} *address*^{f}), it means that the first table is ready to return the values of the *telNo* and the *name* of the subscriber while the second table can give the *address* only if the value of the *telNo* attribute is known.

Regular set of relational operators are insufficient in order to answer queries in the presence of restricted sources. The following subsections demonstrate how two query operators, DAccess and DJoin which are variants of relational scan and relational join operators customized for querying restricted sources, behave.

2.1. Dependent access operator: DAccess

Although we can model the restricted sources with formalization of ‘binding patterns’, due to the access restrictions of the sources we cannot use the query processing operators, like relational scan and relational join. In the example, we have binding patterns of the tables as: *Telephone*(*name*^{f} *telNo*^{f}) and *Address*(*telNo*^{b} *address*^{f}). Binding pattern of the table Address prevents the use of relational scan operator in order to get the address list of the subscribers. In order to get the address value of the subscriber we have to give the values of the telNo attribute. So we need a new scan operator which is able to deal with the restricted sources in the presence of the binding patterns; we name this operator in our study DAccess as D indicates its dependency on the values of the input attribute(s). While the relational scan operator always returns the same result set, this new operator DAccess returns different sets depending on its input set. As seen in Fig. 2 it is a simple operator which accepts a set as its input and returns another set as its output. ‘Formal semantics of DAccess is as follows: Consider a table $R(X^bY^f)$ and χ be a set of values for X. Then, DAccess$(R(X^bY^f))\chi =^{\sigma} X^{\in}\chi R(X, Y, Z)$ [17].

DAccess is a simple operator which is in contact with the wrapper of the restricted source and functions as an adapter to hide the heterogeneity of the query capability of the restricted source. It needs to know


Values of bound attribute(s)
90-212-8978990
789 sok. 43 Etiler ISTANBUL
Tuples corresponding to the values of bound attribute(s)
DAccess Operator
Source Wrapper

telNo	*address*
90-232-8990786	234 sok. 31 Bostanlı IZMIR
90-312-7645673	556 sok. 45 Çankaya ANKARA
90-212-8978990	789 sok. 43 Etiler ISTANBUL
90-232-7506530	786 sok. 1 IZMIR

Fig. 2. Function of DAccess operator.

the bound attribute(s) and output that can be retrieved from the source when the bound attribute(s) is supplied. The wrapper of the source should have corresponding interface to pass the values of the bound attribute(s) to the restricted source and corresponding tuples to the DAccess operator.

2.2. Dependent join operator: DJoin

In this section we will describe a join algorithm designed for restricted sources and based on the hash join algorithm [27] where hash table is created from the smaller relation, R1 (building relation) and hash table is probed with the other relation, R2 (probing relation).

Consider an execution plan that needs a relational join between Telephone and Address tables on their telNo attribute. In relational join both of the following fragments; Join(Telephone, Address) on telNo attribute and Join(Address, Telephone) on telNo attribute are valid since join is commutative. But with restricted sources the second fragment Join(Address, Telephone) on telNo attribute is not valid since the binding pattern of the table Address requires the value of the telNo attribute in order to return the value of the address. So we need a new join operator which is asymmetric in nature, also known as dependent join [5], represented by the $\overrightarrow{\bowtie}$ symbol and which cannot be executed on its own, since the second part of the join requires values to be passed from the execution of the first part. Therefore, we designed a dependent join operator which is able to deal with the restricted sources by cooperating with DAccess operator in internal value passing between the sites, and name it as DJoin. Figure 3 demonstrates the way of coordination between DAccess and DJoin operators. Formal representation of the dependent join is $T \leftarrow Scan(R1(U^fV^f))\overrightarrow{\bowtie}_{V=X} DAccess(R2(X^bY^f))$.

The algorithm of the dependent join, DJoin is presented in Fig. 4. The join operator knows the relations to be joined (R1 and R2) and the relation where the result of the join operation should be stored (T). V and X represent two sets of join attributes from two relations where X contains the bound attributes of R2 which require value passing from V.

If the first relation is not on the site of the DJoin operator, it is retrieved by a scan operator since we know that the first relation has an ff binding pattern. We build a hash table from the first relation (HT1) and at the same time we retrieve the distinct values of the attribute(s) V and store them into the table P. If the join operation is not on the site of R2 (restricted source), we send P to the DAccess operator of the Site2. DAccess operator of Site2 returns the corresponding tuples of R2.

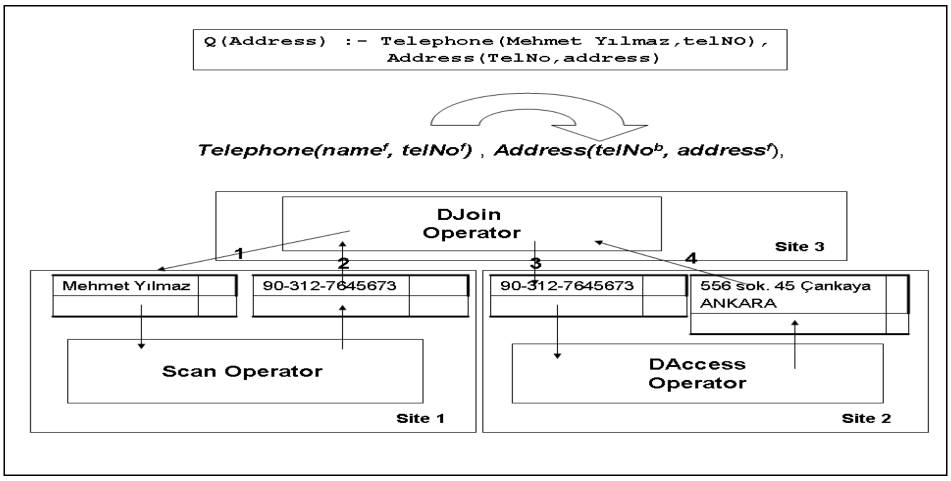

Q(Address) :- Telephone(Mehmet Yılmaz,telNO),
Address(TelNo,address)
Telephone(name^f, telNo^f) , Address(telNo^b, address^f),
DJoin Operator
Site 3
1
2
3
4
Mehmet Yılmaz
90-312-7645673
90-312-7645673
556 sok. 45 Çankaya ANKARA
Scan Operator
DAccess Operator
Site 1
Site 2

Fig. 3. Co-ordination of DJoin and DAccess operators.

```
DJoin (R1, R2, T: Relation; V, X: Set of Attributes)
    {
    If (not local (R1)) then receive (R1);
    Build (R1, HT1);
    P ← Project (R1, V);

    If (not local (R2))    then
                           {
                           send P to S2;
                           // DAccess runs on the remote site
                           receive R2' from S2;
                           }
                           else DAccess(R2, P, R2');

    Probe (HT1, R2', T);
    If (not local (T)) then materialize (T) else send (T);
    }
```

Fig. 4. Algorithm of DJoin.

We assume that on the site of each restricted source there is a DAccess operator: its input is the set of P while its output is subset of the second relation. This specific operator receives set of values corresponding to the bound attribute of the source as input (P), feeding this set to the wrapper of the source retrieves the data corresponding to the free attributes of the source beneath and stores the result in R2'.

3. Mobile join execution model for restricted sources

An important issue in data integration is the use of adaptive or dynamic query optimization techniques to face the high volatility of the web as given in [1,2,7,11,12,22]. From the adaptation point of view of the optimization, they generally try to change the execution plan at run-time, re-optimize query, or use specific operators to deal more flexibly with unpredictable events.

In wide-area distributed system, the join operator should be aware of the inaccuracies of the statistics on relations (e.g., sizes of temporary relations) and the variations in the system state during the execution.

Due to communication cost, the decision to change the execution plan should be taken by the local operator instead of centralized optimizer. Mobile agents with their autonomous and reactive characteristics are used in developing join operators. The join operators of the following sections are mobile agent with the ability of deciding about the place of probing the hash table with the second relation and moving to the place according to their decisions.

3.1. *Mobile dependent join operator: MDJoin*

In [1] an approach for query execution plans by set of reactive join operator agents is proposed. Basically, one agent is in charge of the join; it is initially placed on a site by the optimizer. It s assumed that (i) R1 and R2 are either base or temporary relations, (ii) R1 is located on site S1 and R2 is on site S2, (iii) size of R1 $<$ size of R2, (iv) T=Join(R1, R2) and must be materialized on an indefinite site. First, the mobile join agent builds the hash table from R1 and computes the statistics on R1 simultaneously. Then it decides on its own whether to move or not (on site S2 or on another site) by means of a decision function (i.e. size of R2 $>$ size of R1 $+$ size of T) finds the migration site.

We want to use the similar approach as explained above to design mobile join operators for restricted sources. As in the case of DJoin, mobile join algorithm designed for restricted sources is based on the hash join algorithm [27]. The algorithm of our first mobile dependent join operator, MDJoin, is presented in Fig. 5. Example is the same as the one of DJoin explained in Section 2.2. If the first relation is not on the site of the DJoin operator, it is retrieved by a scan operator since we know that the first relation has an ff binding pattern. We build a hash table from the first relation (HT1). By relational project operation we retrieve the distinct values of the attribute(s) V and store them in the table P just like before. But the difference from static DJoin starts here; after building the hash table we are now able to calculate true statistics of R1and P and we have better idea about the execution environment. MDJoin can now make a decision about where to continue its exeution and migrates to the Site according to its decision function (i.e. i.e. size of R2 $>$ size of R1 $+$ size of T $+$ size of P).

After migration step it is ready to get data from R2; if it is not on the site of R2' then it sends P to the Site2 and receives tuples of R2 corresponding to P in R2' through the DAccess operator of the Site2. In 'Probe (HT1, R2', T)' step, the join operator probes the hash table with R2', and with the last line of the algorithm the result relation is either send to the client site or materialized.

It resembles the semi-join based mobile join algorithm introduced in [1], in a way having an initial projection step on R1. But the difference between the two operators is two-folds: (i) it is not possible to switch to mobile join in some part of MDJoin, and (ii) it is not possible to probe hash table build from P with R2 like in mobile semi-join. We need DAccess operator on the second source due to the restriction imposed by the source.

3.2. *Sampling mobile dependent join operator: SMDJoin*

Although the MDJoin operator, with its mobile and autonomous nature, is capable of joining data retrieved from bound relations and it is more reactive to the changes in run time environment compared to the DJoin operator of the standard strategy. All the information it has about R2 is coming from the optimizer which is compile-time information. We can say that it is blind to the run-time information like data characteristics, data availability and system status concerning Site2. For the cases where the information about the second site is limited or not available, MDJoin can make an improper decision about its location for probing step. To cover such cases we propose to include a mechanism to collect

```
MDJoin (R1, R2, T: Relation; V, X: Set of Attributes)
  {
    If (not local (R1)) then receive (R1);
    Build (R1, HT1);                          // compute statistics on R1
    P ← Project (R1, V);

    Site ← Decision (StatInfo,
                     DataAvailabilitiy,
                     SystemState);
    If (not local (Site) then migrate on Site;   // with P and HT1

    If (not local (R2)) then
        {
            send P to S2;
            //  DAccess runs on the remote site
            receive R2' from S2;
        }
        else
             DAccess(R2, P, R2');

    Probe (HT1, R2', T);
    If (not local (T)) then send (T) else materialize (T);
    }
```

Fig. 5. Algorithm of MDJoin.

information about second site so as to allow the operator make better estimation about the location of the join operation.

SMDJoin operator, as shown in Fig. 6, has a sampling step after building the hash table and projecting (V) from R1 which is stored in P. We make sampling part (between the points P1 and P2) which consists of selecting a sample of data from P (we use ‘p’ to name it), sending it to DAccess operator of Site2 and receiving R2’$_p$ (R2’ corresponding to p). At the point **P1**, size of P and p; from this information it is possible to retrieve sample ratio given as:

Sample Ratio = Size of P / Size of p.

After completing the sampling step, at point P2, we also know the size of R2’$_p$, cost of sending p, cost of computing R2’$_p$ and receiving R2’$_p$. So the estimations for the size of R2’, costs of sending P, preparing R2’ and receiving R2’ can be revised leaning on the new information as shown below:

Size of R2’ = (Size of R2’p $\times$ Sample Ratio) $-$ Size of R2’p
Cost of Sending (P-p) = (Cost of Sending p $\times$ Sample Ratio) $-$ Cost of Sending p
Cost of Computing R2’ = (Cost of Computing R2’p $\times$ Sample Ratio) $-$ Cost of Computing R2’p
Cost of Receiving R2’ = (Cost of Receiving R2’p $\times$ Sample Ratio) $-$ Cost of Receiving R2’p.

So at the point P2, SMDJoin can choose the site of the join execution with enriched information which includes computed size of P and p, revised size R2’, revised cost of sending P, preparing R2’, receiving R2’. Remaining part of the algorithm of SMDJoin is quite similar to the one of MDJoin. While receiving data from R2; P now does not include the set of p since corresponding tuples of R2 are already received in R2’$_p$. In probe step we use R2’$_p$ together with R2’ and the last step is materialization of T.

As a comment on the algorithm; we should say that the size of p has impact on the accuracy of the estimation; if we increase the size, the estimation becomes more accurate. However, large volume of p may result in more overhead with this initial investigation mechanism. This sampling should be done in a way such that the page is a good representative of R1 and R2’$_p$. Estimation accuracy of the selectivity

```
SMDJoin (R1, R2, T: Relation; V, X: Set of Attributes)
  {
    If (not local (R1)) then receive (R1);
    Build (R1, HT1);
    P ← Project (R1, V);                  // compute statistics on R1 and P
    p ← sample(P);                        // select some tuples
P1
    If (not local (R2))  then
                          {
                          send p  to S2;
                          //  DAccess runs on the remote site
                          receive $R2'_p$ from S2;
                          }
                         else
                           DAccess(R2, p, $R2'_p$);
P2                                        // revise statistics on $R2'_p$
    Site ← Decision (StatInfo,
                     DataAvailabilitiy,
                     SystemState);
    If (not local (Site) then migrate on Site;

    If (not local (R2))  then
                          {
                          send (P-p)  to S2;
                          //  DAccess runs on the remote site
                          receive R2' from S2;
                          }
                         else
                           DAccess(R2, (P-p), R2');

    Probe (HT1, (R2'U $R2'_p$), T);
    If (not local (T) then send (T) else materialize (T);
  }
```

Fig. 6. Algorithm of SMDJoin.

factor which is the ratio of R1 tuples corresponding to R2 tuples depends on the accuracy of the decision criteria embedded in sample(P) step.

4. Performance evaluation

In this section, we first explain the experimental environment then make evaluation on the performances of the operator of standard strategy, DJoin and the operators of dynamic decentralized strategy, MDJoin and SMDJoin. The focus of the evaluation is on their performance with respect to the following parameters: response time in the presence of the variations between the compile-time estimations and the run-time computations of the sizes of the relations, speedup[1] realized in comparison with the network bandwidth and CPU frequency. Special emphasis is given to the variation between the compile-time estimation of the optimizer and the run-time computation of the parameters since errors in operator level reveals out propagation effect on the error in the query level [6]. Cost of code transfer is not taken into account (codes are supposed to be resident on mediator sites hosting mobile agents; additionally, it is of little significance compared to the transfer cost of tuples). A convenient mechanism for data serialization without specific overhead is assumed to exist as well.

[1]The Speedup: response time of DJoin / (response time of MDJoin or SMDJoin).

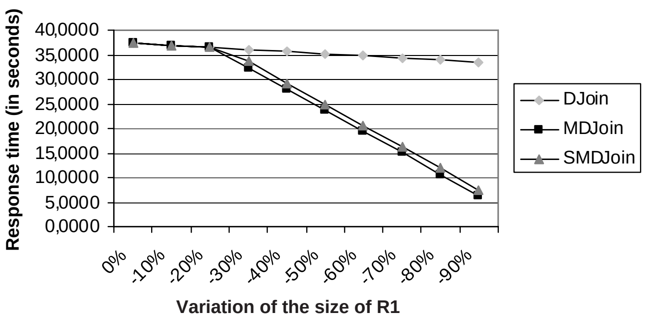


Fig. 7. Response time with decreasing size of R1 (becoming less than the estimation).

4.1. *Experimental environment and simulation model*

Execution model has been simulated by means of a simulator validated by an implementation on a network of workstations [1]. The validation of the simulator consisted in comparing the results obtained by the simulator with those of execution. The simulator allows to test hardware configurations (CPU and network bandwidth) which we do not possess. Simulation model, further hypothesis and parameters are given in Appendix A.

To have quantative comparison, R1 is considered to be a temporary relation (i.e. stemming from two sub-queries) and R2 is considered to be restricted source which can only be accessed with DAccess operator. Their sizes are estimated by the optimizer as 10.000 tuples for R1 (on site S1) and 20.000 tuples (on site S2). The estimated selectivity factor is 1.5/20.000 [29]. Hence estimated size of the result relation is 15.000 tuples. The size of p for SMDJoin operator is 512 tuples, it is kept fixed in all cases since seeing the impact of the size of p is beyond the objective of this paper.

4.2. *Impact of the variation of the sizes of data*

The behaviors of DJoin, MDJoin and SMDJoin are analyzed in the cases of decreasing and increasing size of the data of both sides in order to see the behaviors of the operators when the optimizer under or over-estimates at compile-time.

– Variation of the Size of R1
  In the experiment of Fig. 7, the response time depending on the variation between the compile-time estimation and the run-time computation of the size of R1 is analyzed by decreasing the number of R1 tuples; as a consequence P is decreased. The result relation is stored on the site S1, the optimizer plans to build the hash table on site S1 and to transfer R2' to site S1. DJoin, MDJoin and SMDJoin are located on site S1 (for build phase). Up to -30% of variation DJoin performs slightly better than MDJoin and SMDJoin since mobile operators have some overhead. After -30% of variation between compile-time estimation and run-time computation of the size of R1, MDJoin and SMDJoin are more efficient. From this threshold, the global volume of data to be transferred is less if the join migrates. We see that MDJoin and SMDJoin behave in a similar manner for the cases when the variation of the size of R1 is greater than 30%.
  The impact of the variation between the compile-time estimation and the run-time computation of the size of R1 is evaluated by increasing its size; the size of P is increased as well. The optimizer plans to transfer R1 to the site S2, and the result relation must be stored on another site. In this case the transfer cost of the result relation is not estimated. As seen in Fig. 8, between 10.000 and 20.000

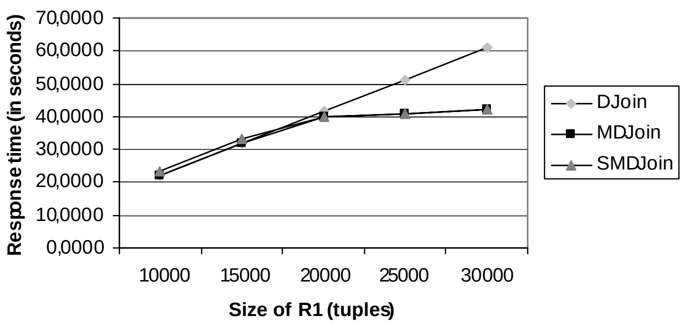


Fig. 8. Response time with increasing size of R1 (exceeding the estimation).

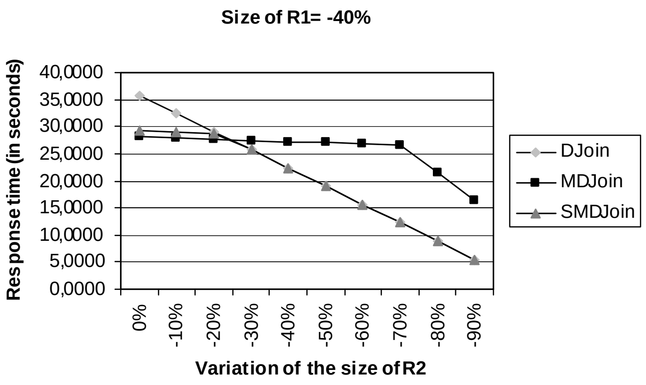


Fig. 9. Response time with decreasing size of R2' (becoming less than the estimation).

tuples of R1, the response time of MDJoin and SMDJoin are slightly higher than DJoin. This is due to the agent migration time from site S1 to S2. While DJoin is located on site S2 at all cases, mobile operators are located on site S1 after the size of R1 exceeds the size of R2' (which is known to be 20.000 tuples). After 20.000 tuples of R1, MDJoin and SMDJoin decide to stay on site S1 and transfer R2', while DJoin continues to transfer R1 to S2. Hence, the MDJoin and SMDJoin reduce the response time by 20% to 40% by minimizing the transferred data volume. Again the behaviors of MDJoin and SMDJoin are quite similar for this case as well.

– Variation of the Size of R1 and R2'

  In Fig. 9 we see the size of R2' becoming less than expected for the case of the variation of the size of R1 is -40%. Up to 30% of variation of the size of R2' both MDJoin and SMDJoin perform slightly better than the DJoin since agent migration brings an overhead. After 30% of variation, it is not a good decision to move so MDJoin (not seeing the variation between the compile-time estimation and the computed size of R2') moves and performs worse than DJoin and SMDJoin. DJoin is located on the correct site by chance. SMDJoin with the help of its sampling step chooses the right site.

  The results observed from Fig. 10 are similar to the above case. Since the variation of the size of R1 now is bigger than the above case, -60%, to choose the right site is even more important. Superiority of SMDJoin becomes more evident since it chooses the right location for all the variations between the compile-time estimation and run-time computations of the sizes of R1 and R2'.

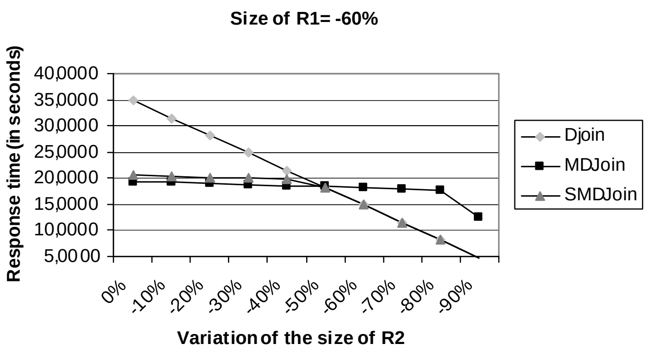


Fig. 10. Response time with decreasing size of R2' (becoming less than the estimation).

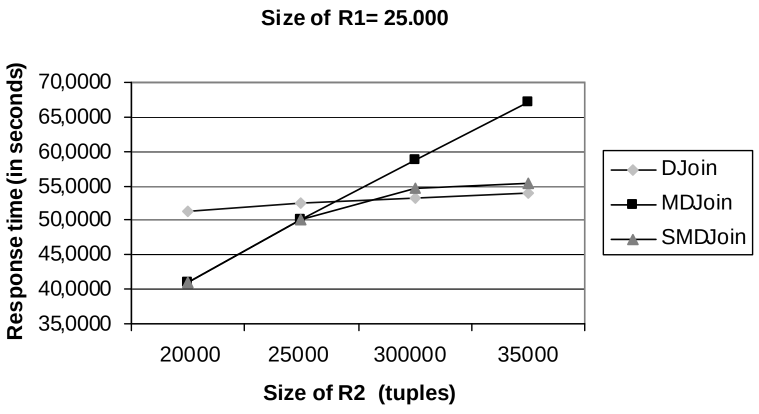


Fig. 11. Response time with increasing size of R2' (exceeding the estimation).

Figure 11 demonstrates the case of R2' being more than expected. We see that DJoin and SMDJoin behave similarly after the size of R2' exceeds the estimated size since DJoin is already on site S2 and SMDJoin sees the variation between the compile-time estimation and run-time computation of the size of R2' and stays on site S2. The performance of MDJoin gets worse since it is blind to the variation of size of R2' and decides to move R2'. When R2' is bigger than expected it is not a good decision to move.

4.3. Impact of the CPU frequency and the communication bandwidth

In this section we analyze the impact of the CPU frequency and the communication bandwidth on speedup realized by the two mobile operators; MDJoin and SMDJoin. Figure 12 shows the speedup of MDJoin for several levels of variations between the compile-time estimation and the run-time computation of the size of the R1. We see that up to 1 GHz there is a speedup. After 1 GHz, speedup is limited by network bandwidth that constitutes a bottleneck.

Figure 13 shows the speedup of SMDJoin for several levels of variations between the compile-time estimation and the run-time computation of the size of R1. Again, we see that up to 1 GHz there is a speedup in relation to CPU frequency. Figure 14 is the comparison of the speedups realized by MDJoin and SMDJoin depending on the CPU frequency when the variation between the estimation of

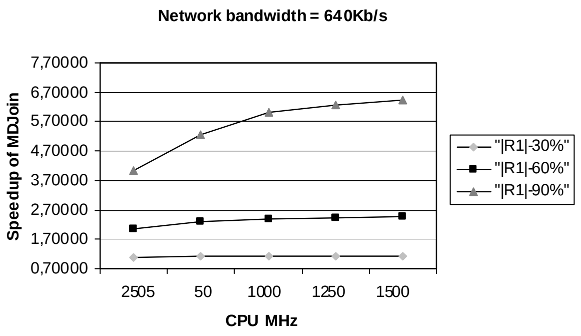


Fig. 12. Speedup of MDJoin with increasing CPU frequency for different variations of the size of R1.

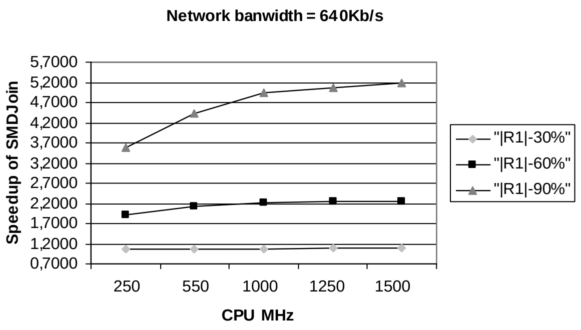


Fig. 13. Speedup of SMDJoin with increasing CPU frequency for different variations on the size of R1.

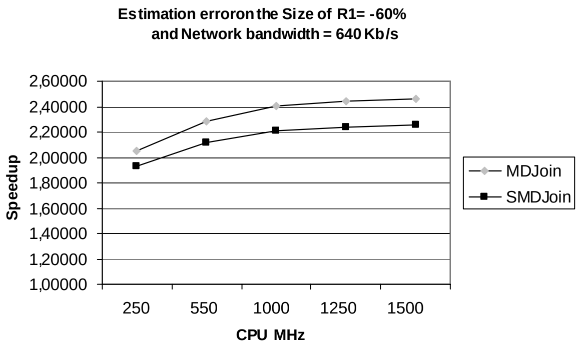


Fig. 14. Speedups of MDJoin and SMDJoin with increasing CPU frequency for variations on the size of R1 $= -60\%$.

the optimizer and the size of R1 $= -60\%$. We see that SMDJoin achieves slightly less speedup due to overhead caused by its sampling step.

Figure 15 shows speedup realized by MDJoin operator for several levels of variation between the compile-time estimation and run-time computation of the size of R1. It can be seen that speedup decreases after 640 Kb/sec to 1.28 Mbs/sec of bandwidth confirming the advantage of mobility for bandwidth values less than 640 Kb/sec which is quite likely to happen in large scale environment.

Figure 16 shows speedups realized by SMDJoin operator for several levels of variation between the estimated and computed size of R1. Results are quite similar to those of the above case.

Figure 17 is the comparison of the speedups of MDJoin and SMDJoin operators depending on the

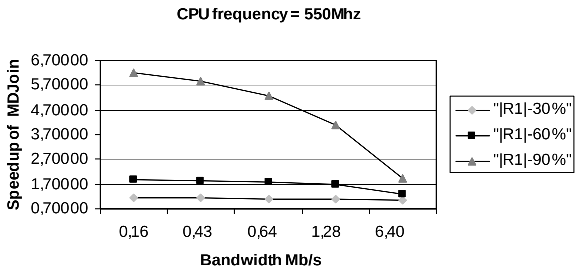


Fig. 15. Speedup of MDJoin with increasing communication bandwidth for different variation of the size of R1.

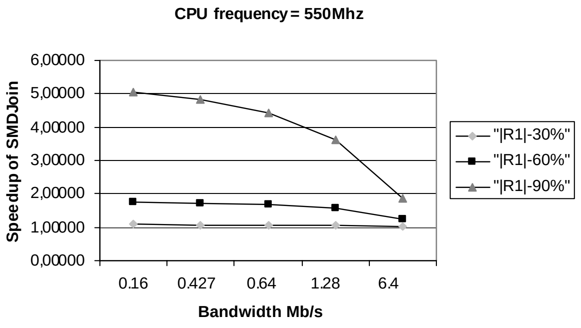


Fig. 16. Speedup of SMDJoin with increasing communication bandwidth for different variation of the size of R1.

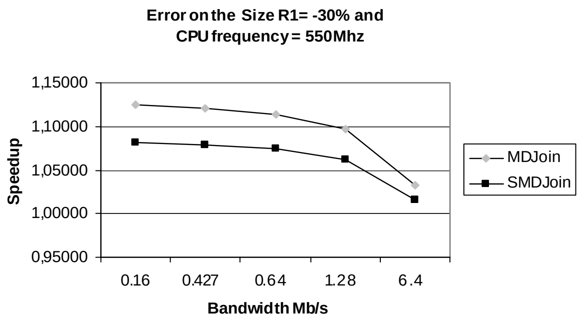


Fig. 17. Speedups of MDJoin and SMDJoin with increasing communication bandwidth for variation of the size of R1 $= -30\%$.

communication bandwidth when the variation of the size of R1 $= -30\%$. We see that SMDJoin brings slightly less speedup due to its sampling step.

5. Discussion on the efficiency of mobile join operators

The results demonstrated that if the size of the data from the first source is more than 30% different than the estimation made by the optimizer, our mobile dependent operators have better performance compared to the operator of the standard strategy. Against the variations of the size of the data coming from the second source SMDJoin is superior to MDJoin, since due to its sampling step it sees the variation of the size of the data of the second source. The overhead of this sampling step for SMDJoin is less than 10%

of increase in response time compared to MDJoin for the cases where they both make the same decision for the location of the probe step. But if the decision for the site to continue the execution is wrong (for MDJoin) response time of MDJoin is higher by up to 4 times depending on the variation between the estimated and computed sizes of R1 and R2’ than the one of SMDJoin.

For measuring the effect of the CPU frequency on the response times of the standard and mobile strategy, we executed our operators with different levels of the variation between the compile-time estimation and run-time computation of the size of the data of the first source. The results demonstrated that there is an increasing speedup generated by the mobile operators up to 1 GHz of CPU frequency. After 1 GHz speedup is limited by network bandwidth that constitutes a bottleneck.

Experiments on the impact of the communication bandwidth were done with similar approach as the ones of CPU frequency and the results show that there is a speedup succeeded by mobile operators up to 640 Kb/sec of communication bandwidth. The advantage of mobility for the cases where the communication bandwidth is probably less than 640 Kb/sec is evident.

Although the response times of our performance evaluation are not the same, the shapes of the most of the curves related with MDJoin are similar to performance results of mobile join operator of [1]. Due to the inadequacy of the scale granularity it is not possible to demonstrate the impact of computing and/or transmitting set of P and retrieving R2’ through DAccess operator. However, it is not wrong to say that the behavior of mobile join of [1] and MDJoin are similar. The behavior of SMDJoin is different than the mobile join of [1] and MDJoin when there is a variation between compile-time estimated and run-time computed data parameters of R2’.

6. Related work

As mentioned in introduction, related work on virtual data integration environment is can be grouped in relation with their emphasis as being on restricted sources or adaptive query processing. The related work whose emphasis is on restricted sources consists of three groups of efforts: (i) modeling the heterogeneity of the sources in terms of access restrictions and query capabilities [18,25], (ii) generating valid, efficient and complete query execution plans in the level of the optimizer [4,13,15,19,21,25] and (iii) developing query execution operators handling restricted access [5,17]. The related work with the emphasis of adaptive query processing can be grouped as centralized [2,7,8,11,12,28] or decentralized [1, 9,10,14,16,20,23,24,26] approaches.

Need for a simple abstraction mechanism in order to hide the heterogeneity of the sources in terms query capabilities and source descriptions is the striving motivation behind the related work focusing on modeling restricted sources. Initially [25] proposed the idea of expressing a given query in terms of the limited query templates that this source can answer. In [18] the concept of table with binding patterns is used to model access restrictions for relational, object-oriented, and XML data sources, and programs.

We see the second group of research work focusing on query optimization with restricted sources. The [25] proposes algorithms for answering queries using views with binding patterns. The [4] considers source restrictions by translating source binding patterns into rules in a datalog program. The [19] introduces the problem of relative containment which formalizes the notion of query containment relative to the sources available to the data integration system and show that relative containment for conjunctive queries is still decidable in the presence of access restrictions even though finding all answers to such queries may require recursive datalog program over the sources. The [15] points out the problem of computing the complete answer to the query using the access patterns permitted by the relations and testing the stability of the various classes of queries including conjunctive queries, unions of conjunctive queries, and conjunctive queries with comparisons. The [13] describes optimization techniques handling recursive query plans that arise commonly in the presence of source restrictions; first, greedy algorithm that minimizes the execution plan removing redundant and overlapping sources without loss of completeness is introduced, second, enhancement of the greedy algorithm to override the default binding patterns that produce high traffic is demonstrated. In [21] the problem of answering queries over sources with limited access patterns is studied; they have extended the results of [15] to conjunctive queries with negation and unions of conjunctive queries with negation.

Third group of research effort is the contributions made to the query execution environment in the presence of restricted sources. The relational operators like scan or join need to be revised and redesigned in order to become suitable to process the query in limited and asymmetric nature of the environment with restricted sources. The [5] demonstrates an approach to combine the query facilities of traditional databases with existing search engines, enabled by two virtual tables and the technique called asynchronous iteration that enables concurrency between query processing and multiple web search requests. The work of [17] proposes a query operator, BindJoin which can exploit a resource with binding patterns and deliver most of its output in the early stages of the execution.

The main idea behind the related work as we classified to be centralized adaptive query processing [2, 7,8,11,12,28] is to change the query execution plan at run-time in order to react to estimation errors (i.e. variation between the parameters estimated at compile-time and the parameters computed at run-time) and resource unavailability (i.e. data, CPU, memory, networks). Related works which can be classified as decentralized approach in adaptive query processing [1,10,20,26] study the contribution of mobile agents to autonomous and self-adaptive query execution in large-scale.

Much research focus on data access from wireless mobile devices [9,14,16,23,24]. introduces an approach for join processing on mobile devices as to reduce transferred data considering the limitations of the mobile devices with the assumption that user query posed on arbitrary non-collaborative data sources [9] is another work concentrating on the impact of using mobile agents in order to reduce the energy consumption of the mobile device and query response time [14] studied join and query processing in wireless environments considering asymmetric features such as the computing capability between fixed and mobile hosts, the transmission bandwidth between fixed and mobile hosts and the transmission cost among local and remote hosts.

In the related work explained above there is no investigation which contributes to the third group of efforts focusing on restricted sources with decentralized adaptive query processing approach implemented by mobile agents for a virtual data integration environment.

7. Conclusion and future work

This paper has demonstrated the use of mobile agents in query execution operators specifically developed for restricted sources. These benefit from autonomous and reactive features of mobile agents in a way to self-adapt dynamically to the variations between the compile-time estimation and run-time computation of the parameters of data. Their ways of decentralized adaptation bring out the advantage of avoiding bottleneck and time overheads commonly associated with centralized monitoring.

The performance study has shown that the mobility was effective (i) when the variation between the compile-time estimated and run-time computed sizes of the data R1and R2', is more than 30%, (ii) the

[2]The duration of agent migration includes serialization of data, transfer of the serialized data and the execution state and de-serialization of data.

Table 1
Simulation parameters

Disk parameters	Disk page size	4 KB
	Average time to read a 4-KB page	47.5 μs
	Average time to write a 4-KB page	195 μs
CPU parameters	Pentium III processor	550 MHz
	Memory size	512 MB
Network parameters	Maximum bucket size	4 KB
	Time to send a 4-KB page	50 ms
	Latency	20 ms
Miscellaneous	Average number of tuples in page	32
	Agent size	2806 B
	Duration of agent migration	150 ms

network bandwidth is less than 640 Kb/s, (iii) the processor frequency is greater than 550 MHz and (iv) SMDJoin has evidently better performance than MDJoin for the cases where there is a variation between the compile-time estimated and run-time computed size of R2'. In large-scale virtual data integration where data sources are expected to be heterogeneous and autonomous; the various data sources do not export needed statistical information or the statistical information is subject to be obsolete. It is difficult for an optimizer to estimate the cost, thus 30% of error in the estimation is not high. In addition, a network bandwidth of 640 Kb/s can often be experienced. So in this context, the contribution of mobility can be taken into consideration. MDJoin can be efficient for the cases where the probability of estimation error on data coming from the second source is low whereas the second operator, SMDJoin is efficient for the cases where the probability of estimation error on data coming from the second source is high.

As a future work we would like to add sociability aspect to our new operators, we would like them co-ordinate for handling complex query execution plans. By this way they will be able to share their knowledge during execution of the query and reach global optimum. The results of the response time of SMDJoin impressed us since it was not blind to the sizes of the data coming from both of the sources it always make good decisions about to place execution. But its decision depends on its sampling step so making this sampling good enough to represent the query can be another future work.

Another point, we find worth of investigation is, in the level of the optimizer, we want to see the effect of the mobility on search space generation. Finally, because of the data source distribution on a large-scale network, it becomes impossible to maintain on every network site a catalog including all the statistics of the base relations, all the characteristics of the site architectures connected on the network and all the cost formulae. Hence, we can point out the importance to define an embedded cost model into the agent in order to guarantee always, its autonomy. These issues form the basis of our current and future research.

Acknowledgements

We would like to thank the anonymous referees for their pertinent and valuable comments.

Appendix A. Simulation Model

Two main modules constitute the simulator: the first one for the simulated method and the second one for the underlying architecture. Disk, CPU, and memory characteristics describe each site of the target architecture. It is assumed that characteristics of all sites (and of all communication links) are the same. Presently, site failures and data unavailability are not considered even if the risk is high in large-scale distributed systems. Finally, it is assumed that agents' buffer size is unlimited. The main costs associated with the basic operations and with the parameters are given in Table 1. Additionally, as the size of the hash table is variable, the simulator computes the duration of the hash table transferred and serialization from the simulation parameters.

References

[1] J.P. Arcangeli, A. Hameurlain, F. Migeon and F. Morvan, Mobile agent based self-adaptive join for wide-area distributed query processing, *Journal of Database Management* **15**(4) (Oct–Dec. 2004), 25–44.

[2] R. Avnur and J. Hellerstein, *Eddies: Continuously Adaptive Query Processing*, in the Proc. of ACM SIGMOD International Conference on Management of Data, Dallas, Texas, USA, May, 2000.

[3] S. Chaudhuri, An overview of query optimization in relational systems, *In the Proc. of ACMSIGMOD* **27** (1998), 34–43, ACM Press.

[4] O.M. Duschka, *Query Planning and Optimization in Information Integration*, Ph.D thesis, Stanford University, Stanford, Calfornia, 1998.

[5] R. Goldman and J. Widom, *WSQ/DSQ: A practical approach for combined querying of databases and the web*, In Proc. of ACM SIGMOD Conf., 2000.

[6] Y.E. Ioannidis and S. Christodoulakis, *On the propagation of errors in the size of join results*, In Proc. of the ACM SIGMOD Intl. Conf. on Management of Data, 1991, 268–277.

[7] Z.G. Ives, D. Florescu and M. Friedman, *An Adaptive Query Execution System for Data Integration*, In the Proc. of ACM SIGMOD Conference, 1999.

[8] Z.G. Ives, Al. Y. Halevy and D.S. Weld, *Adapting to Source Properties in Processing Data Integration Queries*, Proc. of the ACM SIGMOD, June, 2004, 395–406.

[9] Y. Jiao and A.R. Hurson, *Mobile agents in mobile heterogeneous database environment – Performance and power consumption analysis*, In Proc. of the Advanced Simulation Technologies Conference (ASTC), Arlington, Virginia, 2004, 185–190.

[10] R. Jones and J. Brown, *Distributed Query Processing via Mobile Agents*, University of Maryland, from http://www.cs.umd.edu/˜keleher/818s97/ryan/paper.html, Retrieved October 19, 2003.

[11] N. Kabra and D. DeWitt, *Efficient mid-query re-optimization of sub-optimal query execution plans*, Proc. of ACM SIGMOD, 1998, 106–117.

[12] L. Khan, D. Mcleod and C. Shahabi, An Adaptive Probe-based Technique to Optimize Join Queries in Distributed Internet Databases, *Knowledge and Information Systems* **2** (2000), 373–385.

[13] E. Lambrecht, S. Kambhampati and S. Gnanaprakasam, Optimizing recursive information gathering plans, *Journal of Intelligent Information Systems* **22**(2) (March, 2004), 119–153.

[14] C.-H. Lee and M.-S Chen, Processing Distributed Mobile Queries with Interleaved Remote Mobile Joins, *IEEE Transactions on Computers* **51**(10) (2002), 1182–1195.

[15] C. Li, Computing Complete Answers to Queries in the Presence of Limited Access Patterns, *The VLDB Journal* **12** (2003), 211–227.

[16] E. Lo, N. Mamoulis, D.W. Cheung, W.S. Ho and P. Kalnis, *Processing ad-hoc joins on mobile devices*, Proc. of the International Conference on Database and Expert Systems, 2004, 611–621.

[17] I. Manolescu, L. Bouganim, F. Fabret and E. Simon, *Efficient querying of distributed resources in mediator systems*, In the Proc. of the Confederated International Conferences DOA, CoopIS and ODBASE, LNCS 2519, Springer-Verlag, 2002, 468–485.

[18] I. Manolescu, *Optimization techniques for integration of heterogeneous and distributed data sources*, Ph.D thesis, Universty of Versailles, 2001.

[19] T. Millstein, A. Halevy and M. Friedman, Query containment for data integration systems, *Journal of Computer and System Sciences* **66** (2003), 20–39.

[20] F. Morvan, M. Hussein and A. Hameurlain, *Mobile Agent Cooperation Methods for Large Scale Distributed Dynamic Query Optimization*, Fifth Intl. Workshop on Parallel and Distributed Databases: Innovative Applications and New Architectures (PaDD03), Prague, 1–5 September, 2003.

[21] A. Nash and B. Ludascher, *Processing unions of conjunctive queries with negation under limited access patterns*, LNCS, 2004, 422-440.

[22] M. Ouzzani and A. Bouguettaya, Query Processing and Optimization on the Web, *Distributed and Parallel Databases* **15** (2004), 187–218.

[23] S. Papastavrou, G. Samaras and E. Pitoura, Mobile agents for www distributed database access, *Transaction Knowledge and Data Engineering* **12**(5) (September/October, 2000).
[24] W-C. Peng and M-S. Chen, *Exploiting the features of asymmetry for query processing in a mobile computing environment*, In the Proc. of Fifth IFCIS Conference on Cooperative Information Systems (CoopIS'2000), September 6–8, 2000.
[25] A. Rajaraman, Y. Sagiv and J. Ullman, *Answering queries using templates with binding patterns*, In the Proc. of ACM PODS, San Jose, CA, 1995.
[26] A. Sahuguet and B.C. Pierce and V. Tannen, *Distributed Query Optimization: Can Mobile Agents Help?* http://db.cis.upenn.edu/DL/mobile optimization.pdf, 2000.
[27] D.A. Schneider and D.J. DeWitt, *Tradeoffs in processing complex join queries via hashing in multiprocessor database machines*, In the Proc. of the 16th VLDB Conference Brisbane, Australia, 1990.
[28] C. Shahabi, L. Khan and D. McLeod, A probe-based technique to optimize join queries in distributed internet databases, *Knowledge and Information Systems* **2** (2000), 373–385.
[29] E.J. Shekita, H.C. Young and K.L. Tan, *Multi-join optimization for symmetric multiprocessors*, In the Proc. of 19th Int. Conference on VLDB, 1993, 479–492.
[30] G. Wiederhold, Mediators in the architecture of future information systems, *IEEE Transactions on Computers* **25**(3) (1992), 38–44.

Belgin Ozakar is currently working as an instructor in the department of Computer Engineering of Izmir Institute of Technology. She is doing her phD research based on the joint phD agreement between Aegean University and Paul Sabatier University. She worked at IT industry for 17 years before she joint academia. Her main research interest is optimization in virtual data integration, mobile agent, mobile computing and web mining.

Franck Morvan received a phD degree in Computer Science from Paul Sabatier University in 1994. He worked at Dassault Data Services society for 3 years before he joined Paul Sabatier University. He is currently associate professor and member of the Institute of Research in Computer Science of Toulouse (IRIT). His main reserach interest are optimization in distibuted and parallel databases, mobile agent and mobile computer.

Abdelkader Hameurlain is professor in Computer Science at Paul Sabatier University. He is a member of the Institute of Research in Computer Science of Toulouse (IRIT). His current research interests are in query optimization in parallel and large scale distributed environment, mobile databases, and databases performance. Prof. Hameurlain has been the general chair of the Interational Coference on Database and Expert System Applications (DEXA'02). He was guest editor of two special issues of journals on “Mobile Code and Agent” and “Mobile Databases”.