Robust Placement of Mobile Relational Operators for Large Scale Distributed Query Optimization

Belgin Ergenç[*], Franck Morvan[**], Abdelkader Hameurlain[**]
[*] Izmir Institute of Technology, Urla, Izmir, Turkey
E-mail: belginergenc@iyte.edu.tr
[**] IRIT Laboratory, Université Paul Sabatier, 118, Route de Narbonne, 31062 Toulouse Cedex, France
E-mail: {morvan, hameur}@irit.fr

Abstract

This paper presents a compile-time placement method of mobile relational operators MROs in a large scale environment. MROs are self adaptive to changing run-time conditions by deciding their execution place if they discover compile-time estimation errors. Proposed placement methods tend to have a main drawback with MROs running over a large scale environment: their focus is on finding optimal performance depending on single-point estimation at compile-time, instead of optimal performance over an estimation interval. We propose: (i) to determine the migration space of a MRO including the sites on which the MRO is allowed to migrate during its execution, and (ii) to find the robust site which will allow acceptable response time in an estimation interval. Performance study shows that, with a risk of loosing around 6% in response time, it is possible to gain up to 300% with the proposed robust placement.

1. Introduction

The mediation systems based on the mediator-wrapper architecture [39] manipulate large heterogeneous distributed data sources. One of the major challenges is to propose optimization methods to enable querying efficiently in the presence of unpredictable sources and heavy communication costs. Traditional query processing methods end up with sub-optimal execution plans in such environments since it is difficult to maintain up-to-date statistics and the statistics describing the data coming from the sources and the formulae associated with the cost of operations are not revealed by all the autonomous sources [9, 18, 27]. By the motivation of avoiding performance penalty caused by the optimizer's mistakes in unpredictable and volatile environment, there has been growing number of adaptive query processing approaches that try to re-optimize the execution plan during execution in order to react to the compile-time estimation errors [1, 19, 20]. The adaptation methods may vary but the main behavior is to optimize at compile-time and to execute-monitor-re-optimize at run-time where the monitoring can be centralized or decentralized.

Centralized monitoring does not allow all adaptation methods to be applied on large scale because of the relatively significant message passing on a network with low bandwidth and strong latency, and the bottleneck generated by the optimizer. On the other hand, we observe decentralized methods [8, 19, 37] that try to improve the cost of local processing by adapting the use of the CPU, I/O and memory to the changes in execution environment. Whereas [1, 30] take into account the cost of local processing together with network resources. The approach presented in [1, 30] proposes making mobile every relational operator of an execution plan. A mobile relational operator (MRO) is able to react in an autonomous and decentralized way during its execution with respect to system state or variations in compile-time estimations. In addition, the MRO can migrate from one site to another site [1, 30]. In [1, 30] the control and modifications of execution plans are decentralized since the MRO makes its decision by itself.

Re-optimization methods suffer mainly from re-optimization unaware optimization at compile-time. Decisions made at compile-time can handicap the re-optimization method performance at run-time. For example, the optimizer might decide to place a MRO on a site with compile-time estimations. This placement can obstruct the MRO to move at run-time with refreshed estimations when there is an estimation error. Studies capturing the problem of building bridge between compile-time optimization methods and run-time adaptive methods are critical at this point. The approach presented as proactive re-optimization in [5] is an elegant example of such studies. The authors are trying to find robust execution plans within bounding

0-7695-3049-4/07 $25.00 © 2007 IEEE
DOI 10.1109/.53

boxes around estimates that define the uncertainty in estimates of statistics. They prefer an execution plan with acceptable performance instead of a good plan based on single-point estimations since it might have catastrophic performance in cases of estimation errors. Although, their approach might be worth in a centralized environment, large scale environment imposes paying attention to the aspects of minimization of data transfer in the presence of low bandwidth and strong latency. Hence, the placement of relational operators plays an important role. When we look at the placement methods provided by the parallel and distributed query processing field [6, 7, 10, 11, 13, 15, 23, 25, 32, 41] are based on single-point estimations. Such a placement is as good as compile-time estimations but it yields bad performance in cases of estimation errors. When we use a MRO, its placement is critical since wrong placement can block its adaptation capability. Therefore, we suggest focusing, in this paper, on an initial placement for MROs to insure their adaptation capability at run-time in cases of estimation errors. The challenge is not in finding good placement based on single-point estimates but on defining acceptable placement for an estimation interval in order to avoid dramatic performance at run-time. Our proposal can be seen as an extension of the method proposed by [5] in a large scale context and decentralized environment. The main components of our initial placement method are: (i) migration space which includes the possible execution sites for the MRO, (ii) robust site which permits the discovery of the initial site providing acceptable performance for an estimation interval.

The rest of the paper is organized as follows. Section 2 discusses the problems of single-point placement methods with respect to MRO mapping onto the sites and describes the problem position. Section 3 introduces the components required for initial placement of MROs. Section 4 points out the results of our performance evaluation, based on single join and multi-join. Section 5 is dedicated to the survey of related work. Finally, we give our conclusion.

2. Context and problem position

In this section we will present a mobile execution model [1] in order to detail the problem related with the initial placement of MROs.

2.1. Mobile execution model

On each site participating in the query processing, there is a mediator as presented in [14]. The query is submitted to the mediator where it is transformed and optimized. The operators of an execution plan are executed either by the mediators or by the wrappers. Every MRO is executed on a mediator and a scan operator (fixed operator) is executed on wrapper. On this one, the translation of the sub-query and re-formatting of the result for the mediator will be abstracted by means of a scan operation. Now, we will recall the mobile hash join algorithm [1].

Let us consider an example to illustrate the mobile execution model: R1 and R2 are base or temporary relations, with R1 located on site S_1 and R2 located on site S_2, and $|R1| < |R2|$ ($|R1|$ is the size of R1). T is the result of join between R1 and R2, $T \leftarrow \text{Join}(R1, R2)$, which must be materialized on site S_1 where T is expected. The hash join algorithm[33] comprises 2 steps: in the first step, the hash table is built from the smallest relation R1; in the second step the hash table is probed with the other relation R2. During the building of the hash table, statistics on R1 can be computed precisely. From these statistics on R1 and compile-time statistics on R2, the selectivity factor for example can be revised and statistics on T consequently can be refreshed. Then, the mobile join operator (MJO) decides on its own whether to move or not (on S_2 or on another site) by means of decision function that computes the migration site. The decision function is based on a cost model [14] extended with the mobility cost and have parameters StatInfo (i.e. updated statistics on relations), DataAvailability (i.e. waiting for the first tuple and waiting time for any tuple) and SystemState (i.e. communication bandwidth, CPU, memory capacity). Hash table is moved along with the MJO. For example T is expected on S_1 and compile-time estimate of the optimizer is $|R2| < |R1| + |T|$: consequently the MJO is placed on S_1. However if it appears that $|R2| > |R1| + |T|$ after building the hash table on S_1 the MJO migrates to site S_2 for probing. The algorithm given in Figure 1 illustrates the behavior of the MJO.

2.2. Limitations of single-point placement methods

Present single-point placement methods [10, 23, 32, 41] try to minimize the communication cost by placing the relational operators on the sites minimizing the data transfer by taking into account single-point estimates instead of an estimation interval. Let us demonstrate the problem with a simple query: $T \leftarrow \text{Join}(R1, R2)$, where R1 and R2 are base or temporary relations residing on the sites S_1 and S_2 and the site of T is S_1. Suppose the optimizer decides to place initially the MJO on S_2 with the estimation $|R2| > |R1| + |T|$. If the compile-time estimate of the optimizer is correct, the response time of the query on S_2 is minimized. However, if the compile-time estimate is not correct and for example $|R1| + |T| > |R2|$, in this situation, the MJO loses possibility of migration because its migration triggers the transfer of its hash table and R2. Our simple test shows (Figure 2) that in such situation, the increase in the response time of the MJO might be more than 220% compared with the best placement which is on site S_1. On the other hand, if the optimizer places the MJO on S_1 and if the compile-time estimate is correct, the MJO response time is more than expected with the overhead of migration (the cost of serialization, transfer and de-serialization of the MJO) which is shown around 6% of increase in response time [1, 30]. However, if the compile-time estimate is not correct, MJO can decide to stay on S_1 and has better performance. This simple example shows, with a risk of loosing 6% in response time, it is possible to gain more than 220% in cases of estimation errors.

```
Step 1: if (not local(R1)) then receive(R1);
        Build(R1, HT1);  //and compute the statistics on R1
        Site ← Decision(StatisticalInformation,
                        DataAvailibility, SystemState);
        if (not local(Site)) then migrate on Site;
Step 2: if (not local(R2)) then receive(R2);
        Probe(HT1, R2, T);
        if (not local(T)) then send(T) else materialize(T)
```

Figure 1. Algorithm of the mobile hash join [1]

3. Robust placement

In this section we will first introduce the components of robust placement method and then explain a robust placement algorithm. Robust placement method is based on two components: migration space and robust site.

3.1. Migration space

It is not reasonable to allow a MRO to migrate to all existing sites over large scale environment. The time to compute the better site among all the sites present on the network would be prohibitive. So, the optimizer should define a subset of sites on which the MRO can migrate during its execution: migration space. This one should be defined taking into account the migration spaces of the communicating operators.

Migration space of a MRO is defined as the set of execution sites that the MRO can migrate during its execution. Migration space should include the sites of the producer and consumer operators [10, 25]. Consider the previous example and assume that R1 and R2 are on the sites S_1 and S_2 respectively and the result of the query is expected on site S_1. Hence, the producer operators like Scan(R1) and Scan(R2) should be on the sites S_1 and S_2 respectively and the consumer operator is on the site S_1. Then, the migration space MS of MRO is quoted $MS_{MRO}=\{S_1, S_2\}$.

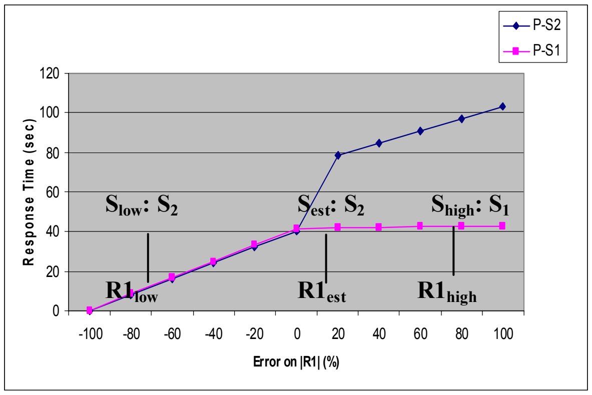

Figure 2. Response times of P-S1 and P-S2 with respect to error on |R1|

3.2. Robust site

Figure 2 shows the response times of two execution plans of the same MJO with respect to the error on |R1|. P-S1 is the plan in which MJO starts its execution on site S_1. P-S2 is the plan in which MJO starts its execution on site S_2. In both plans MJO continues its execution on site S_1 or S_2. Migration space of the MJO is defined as $MS_{MJO}=\{S_1, S_2\}$. We assume to have three points defined by the optimizer corresponding to the estimated size of the relation R1: $R1_{low}$, $R1_{est}$ and $R1_{high}$ represent respectively the cases of smallest, average and highest estimations of the optimizer on |R1|. Inside the interval $[R1_{low}, R1_{est}]$ both plans P-S1 and P-S2 achieve close response times to each other. However, P-S1 incurs a slight extra cost due to migration overhead-threshold which is found around 6% increase in response time [1] in our case. For the interval $[R1_{est}, R1_{high}]$, P-S1 is not affected by the increased size of R1. On the other hand, if we look at the cost of P-S2, we remark that the response time of P-S2 increases dramatically since its migration causes transfer of hash table and R2. As a result, for the interval $[R1_{low}, R1_{est}]$, S_2 is the site giving minimum response time ($S_{low} = S_2$), for $R1_{est}$, S_2 is the site giving minimum response time ($S_{est} = S_2$) and for $[R1_{est}, R1_{high}]$, S_1 gives the minimum response time ($S_{high} = S_1$).

This simple example demonstrates that the execution plans are quite related with the initial placement and not all placements provide acceptable performance over an estimation interval. In order to find a site giving acceptable performance over an estimation interval we should check the response time of the MRO for all the points of the estimation interval. In our case, response times of P-S1 for $R1_{low}$ and $R1_{est}$ is closed to minimum with the threshold, and for $R1_{high}$, PS-1 is the plan with a minimum response time. Hence, the site S1 provides acceptable performance over an estimation interval. We say that it is a robust site for MRO.

In order to formalize the definition of a robust site, we associate to each MRO the following annotation: (i) three estimation points $R1_{low}$, $R1_{est}$ and $R1_{high}$, (ii) a migration space defined as $MS_{MRO}=\{S_1,S_2,\ldots,S_n\}$, (iii) S_{low}, S_{est} and S_{high} corresponding to the sites minimizing the response time corresponding the values of $R1_{low}$, $R1_{est}$ and $R1_{high}$ respectively, (iv) a response time set $=\{RT\ (S_i,\ R1_k)\}$ meaning that the response time achieved by initially placing MRO on S_i with the estimated size of $R1_k$, for $k \in$ {low, est, high} and $S_i \in MS_{MRO}$ and (v) mobility overhead constant of the optimizer defined as threshold.

*A site S_i is said to robust iff: $\forall k \in$ {low, est, high} and $\forall RT(S_i, R1_k) <= RT(S_k, R1_k)$ * threshold.*

A site is robust if and only if, for all estimated points of R1, the response time of MRO initially placed on it, is equal to or close to the response time of the site giving minimum response time for this R1 estimation with the threshold. In other words, MRO should have a response time that is close to the response time of the site giving minimum response time for the all the estimated points in the interval. If we go back to the situation of S_1 and S_2 of Figure 2 we can say that S_1 is robust site since its response time for $R1_{low}$, $R1_{est}$ and $R1_{high}$ is equal or close to the minimum response times of S_{low}, S_{est} and S_{high} with the threshold.

The algorithm of the robust site function given in Figure 3, takes a Node (MRO or Scan operator) as an input and returns a robust site. The robustness of each site is controlled with the CheckRobustness function which takes a site, a MRO and a threshold as input and returns true only if the response times of the MRO for $R1_{low}$, $R1_{est}$ and $R1_{high}$ on the site are equal or close to the response times of S_{low}, S_{est} and S_{high} with the threshold respectively. If S_{low} and S_{high} do not satisfy the requirements of robustness than the site is S_{est}.

```
CheckRobustness(site, Node, threshold): Boolean {
    // threshold is the mobility overhead
Return (RT(site, Node.$R1_{low}$) ≤ RT(Node.$site_{low}$,
        Node.$R1_{low}$) * threshold and
            RT(site, Node.$R1_{est}$) ≤ RT(Node.$site_{est}$,
            Node.$R1_{est}$) * threshold and
            RT(site, Node.$R1_{high}$) ≤ RT(Node.$site_{high}$,
                    Node.$R1_{high}$) * threshold)}

RobustSite(Node, threshold) {
    CheckRobustness(site, Node, threshold): Boolean;
    // SiteMinRT function returns the site giving the
    // minimum response time with respect to given
    // estimation point
    Node.$site_{low}$   ← SiteMinRT(Node,
                    Node.migrationSpace, Node.$R1_{low}$ );
    Node.$site_{est}$   ← SiteMinRT(Node,
                    Node.migrationSpace, Node.$R1_{est}$);
    Node.$site_{high}$  ← SiteMinRT(Node,
                    Node.migrationSpace, Node.$R1_{high}$);
    if   (CheckRobustness(Node.$site_{low}$, Node,
            threshold)) then return Node.$site_{low}$
    // $site_{low}$ : robust site
    else  if (CheckRobustness(Node.$site_{high}$,
            Node, threshold))
                    then return Node.$site_{high}$
    // $site_{high}$ : robust site
                    else return Node.$site_{est}$;
}
```

Figure 3. Robust site algorithm

3.3. Robust placement of mobile relational operators of a query

The robust placement algorithm, Figure 4, places a MRO together with its children. It has as input a MRO, the site where the query result is expected and a threshold. It is a recursive algorithm which places first the right and left child of MRO.

If the Node is a Scan operator, the placement space is defined as the sites where its operand is replicated. It is mapped on a site of the placement space providing the required data with a minimum response time. For a filter operator (i.e. Select or Project operator), its migration space is defined to be the migration space of the direct child operator and it is mapped on the placement site of the direct child operator. When the operator is binary operator (e.g. join), the placement of the right child and left child is done first. Then the migration space of the binary operator is defined and finally the robust site is decided by the RobustSite function among the sites present in its migration space.

```
RobustPlacement(Node, resultSite, threshold){
if (isScan?(Node)) then  {
     Node.migrationSpace ← Node.R1.Replicates};
     Node.site ← SiteMinRT(Node,
                         Node.migrationSpace, Node.R1_est);}
if (isFilter?(Node)) then  {
   RobustPlacement(LeftChild(Node), resultSite,
                         threshold);
   Node.migrationSpace ←
                         LeftChild(Node). migrationSpace;
   Node.site ← LeftChild(Node).site;}
if (isBinary?(Node)) then
     {RobustPlacement(RightChild(Node),  resultSite,
                                       threshold);
   RobustPlacement(LeftChild(Node), resultSite,
            threshold);
   Node.migrationSpace ←
         RightChild(Node).migrationSpace ∪
         Leftchild(Node).migrationSpace ∪ resultSite;
   Node.site ← RobustSite(Node, threshold);}
}
```

Figure 4. Robust placement algorithm

4. Performance evaluation

In this section, we compared single-point and robust placement methods mapping single join and multi-join. We study the behavior of both methods when there is error on the estimations related to the size of relations. Performance study is based on a simulation model which was validated by an implementation on a workstation network [28].

4.1. Simulation model

Simulator is composed of two parts: simulated method and architecture. Disk, CPU, and memory characteristics describe each site of the target architecture. We assume that the characteristics of all the sites and communication links are the same. In addition, site failures and data unavailability are not considered even if the risk is high in large scale distributed systems. Finally, it is assumed that the buffer size of a MRO is unlimited.

The main costs associated with the basic operations and with the parameters are given in Table 1, according to [12, 28] for the approximation of the duration of MRO migration. As the size of the hash table is varying, the simulator computes the duration of the hash table serialization from simulation parameters.

4.2. Single join

Let us consider two relations R1 and R2 residing respectively on sites S_1 and S_2 with number of tuples estimated as 10.000 and 30.000. Also, let us assume a simple hash join J: T = Join(R1, R2). The result is expected on the site S_1. Migration space of the join MSJ= $\{S_1, S_2\}$. The estimated selectivity factor [34] of join J is 1.5/max(‖R1‖, ‖R2‖) where ‖R1‖ and ‖R2‖ indicate the number of tuples. The expected threshold of the optimizer capturing mobility overhead is 1.06 increases in response time of the join operation. Our performance evaluation measures the influence of the errors on the |R1| (i.e. size of R1) on response time with single-point and robust placement methods. The size of R1 is varied from -80%, (80% smaller than the size estimated at compile-time) to 160%, (the size is 160% bigger than the size estimated at compile-time).

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 size of a tuple	128 B
	MRO size	2806 B
	Duration of MRO migration[1]	150 ms

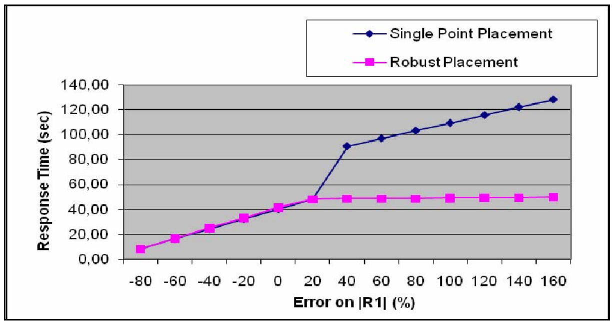

Figure 5. Response time with respect to error on |R1|

Figure 5 demonstrates the impact of the error on |R1| on response time of the join operation. Single-point placement method places the MJO on site S_2 since estimated |R2| > |R1| + |T| whereas robust placement method places the MJO on site S_1 since S_1 is the robust site for $R1_{low}$, $R1_{est}$ and $R1_{high}$ which are estimated by the optimizer as 256 KB, 1 280 KB and 3 328 KB respectively. We observe clearly that the response time of robust placement is higher than the response time of single-point placement for the range

1 The duration of mobile relational operator migration includes serialization of data, transfer of the serialized data and the execution state and de-serialization of data

[-80%, 20%]. In this range, the difference between two response times is between 2,9% and 3,7% which is less than the threshold. On the other hand, for the range of [20%, 160%], the speed up achieved by robust placement method reaches up to 300 % when compared with the response time of single-point placement method. So with the risk of loosing 3.7% in response time, it is possible to increase the performance by 300% approximately.

4.3. Multi-join

Let us consider four relations R1, R2, R3 and R4 residing respectively on sites S_1, S_2, S_3 and S_4 with number of tuples estimated as 10.000, 20.000, 20.000 and 25.000 respectively. We assume a query T = J3(J1(R1,R2), J2(R3,R4)) where the execution plan is a bushy tree. The query result is expected on the site S_5. The migration spaces of joins are $MS_{J1}= \{S_1, S_2, S_5\}$ and $MS_{J2}= \{S_3, S_4, S_5\}$ and $MS_{J3}= \{S_1, S_2, S_3, S_4, S_5\}$. The selectivity factors of joins J1, J2 and J3 are $1.5/\max(\|R_i\|, \|R_j\|)$. The plan generated by the single-point optimizer is to place J1 on the site S_2 since $|R_2| > |R_1|$, J2 on the site S_4 since $|R_4| > |R_3|$. On the other hand, robust placement maps the J1 onto the site S_1 regarding the values of $R1_{low}$, $R1_{est}$ and $R1_{high}$ which are estimated by the optimizer as 256 KB, 1 280 KB and 3 328 KB respectively. J2 is placed on S_3 by checking the robustness of the site for the values of $R3_{low}$, $R3_{est}$ and $R3_{high}$ given as 512 KB, 2 560 KB and 7 168 KB respectively. Placement of J3 is done on S4 by both methods. In order to see the impact of the placement method on the response time of the query we compared both methods in the cases of error on the size of the relations in the query; R1, R2, R3 and R4. We generated error on the size of the relation in a range [-80%, 160%] as in the case of single join.

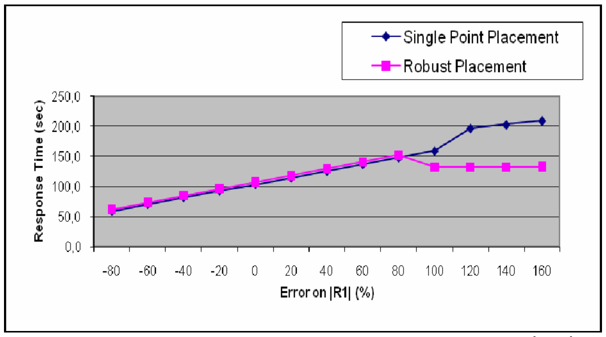

Figure 6. Response time with respect to error on |R1|

Figure 6 demonstrates the impact of the error on |R1| on response time of the query. For the error in the range [-80%, 80%] MJOs placed by single-point placement method remain on their initial sites and the query response time is kept at minimum. On the other hand, if we look at the performance of robust placement method for the same range of error we find that the query response time is higher by [2,9%, 3,43%] since there is mobility overhead caused by J1 and J2. The overhead of each join is less than the threshold. For the range of [80%, 160%], first join placed by robust placement decides to stay on S1 whereas it decides to migrate to S_1 by single-point placement. We remark that robust placement method causes speedup from 16,8% to 36,4% corresponding to the range of [80%, 160%].

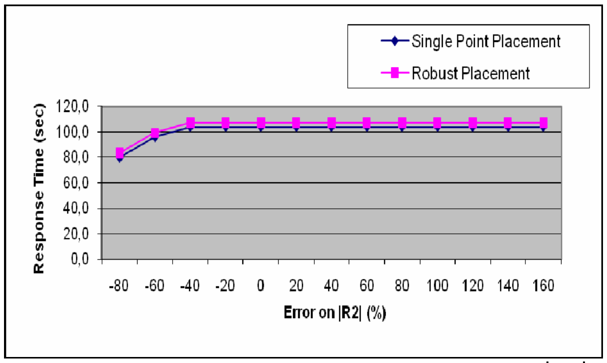

Figure 7. Response time with respect to error on |R2|

Figure 7 demonstrates the impact of the error on |R2| estimated at compile-time on the query response time with the two placement methods. Response time achieved by robust placement is higher by [3,4%, 4,4%]. Changing size of the second relation does not affect the response time of the query in both methods. Since the MJOs do not take any adaptation decision depending on R2. MJOs placed by single-point placement method remain on their initial sites whereas, the first and second MJOs placed by robust placement method move to the sites S_2 and S_4 respectively. Performance of robust placement method is higher than single-point placement since the response times of the first and second MJOs include mobility overhead as well.

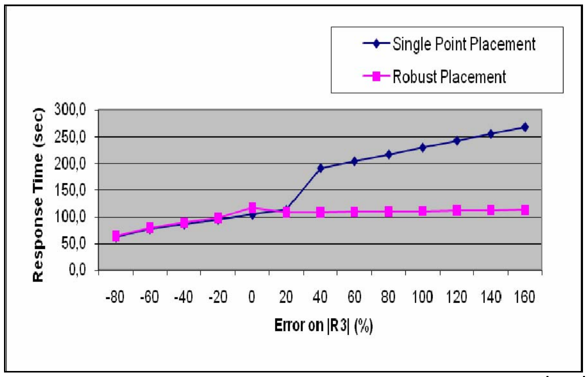

Figure 8. Response time with respect to error on |R3|

Figure 8 shows the impact of the error on |R3| on the query response time. For the range [-80%, 20%], MJOs placed by single-point placement method remain on their initial sites and the query response time is at minimum. For the same range of error, response time of the single-point placement is higher by [2,9%, 5,6%]. On the other hand, for the range [20%, 160%], single-point placement causes an unexpected increase in response time since J2 decides to migrate to site S_3. If we look at the performance of robust placement method, for the range of [-80%, 20%] we find speedup by [4,6%, 57,97%] since J3 decides to stay on site S3.

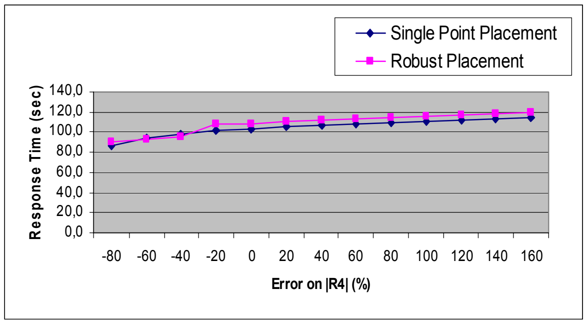

Figure 9. Response time with respect to error on |R4|

Figure 9 demonstrates the impact of the error on |R4| on the query response time with the two placement methods. Performance of robust placement method is less than the single-point placement by [4,1%, 5,6%]. Again, like the case shown in Figure 7, changing |R4| does not make major change in the query response time in both methods since the MJOs do not take any adaptation decision depending on the |R4|.

4.4. Discussion

In this section, we analyzed the performance of single-point placement and robust placement methods when there is error on the compile-time estimation.

In the first part, we analyzed the response time of a single join placed by single-point and robust placement methods when there is an error on |R1| over a range [-80%, 160%]. The result of the study showed that robust placement method outperformed by almost up to a factor of three when the error on |R1| exceeds 20%. For the error less than 20%, the robust placement performance is higher than single-point placement by [2,9%, 3.7%]. This loss is less then the threshold of mobility overhead which is 6%.

In the second part, we analyzed the response time of an execution plan composed of three joins in the form of a bushy tree with the MJOs placed by the single-point and robust placement methods. Again, the error generated on |R1|, |R2|, |R3| and |R4| is between [-80%, 160%]. Robust placement has better performance in case of the error on |R1| when the error exceeds 80% and in case of the error on |R3| when the error exceeds 20%. The performance gain reaches up to 60% in response time. Performance loss the single-point placement in cases of error on |R2| and |R4| estimated at compile-time does not exceed threshold (e.g. 5,6% maximum). We noticed that the error on the size estimated at compile-time of the first relation has greater impact on the query response time. Robust placement method outperforms when there is error on the size of the first relation since it expects adaptation and chooses the initial site according to this possibility.

5. Related work

Large scale environment needs paying attention to two major aspects: heterogeneity and autonomy of the data sources and the minimization of data transfer in presence of low bandwidth and strong latency of network. To overcome the execution plan sub-optimality, various adaptive query processing methods are proposed. The main idea is to use the execution plan selected by the optimizer at compile-time and make re-optimization at run-time to adapt to changing conditions [1, 2, 19, 21]. The adaptive methods can be classified according to nature of the decision making about adaptation: centralized and decentralized methods. In the centralized methods [2, 19, 20, 21, 41], a main process is in charge of monitoring and changing, at runtime, the execution plans, whereas correcting sub-optimality is done in operator or sub-query level in decentralized methods [1, 8, 29, 30, 37].

Regardless of the nature of the decision making about adaptation, the approach of compile-time optimization followed by run-time adaptation has limitations. In fact, the optimizer selects execution plans without taking into consideration the issues affecting adaptation at run-time. The approaches for foreseeing the need for adaptation to changing run-time parameters start with the parametric optimization [16, 17, 20]. The goal is identify several execution plans at compile-time for different ranges of values of run-time parameters and to enable the optimizer to defer the choice of an execution plan at run-time. In error-aware-optimization proposed by [38], intervals are estimates considered to identify robust plans with single uncertain statistics and single join. However, both parametric optimization and error-aware optimization do not consider re-optimization or statistical collection at run-time. Studies presented in [3, 4, 5, 27] are worth considering in terms of narrowing the space between compile-time and run-time re-optimization. They combine and exploit the ideas of switchable plan, switch operator, robust plan at compile-time and collection of statistics and re-optimization at run-time.

The main idea behind these works is, instead of having good plan based on single-point estimations that might have bad performance in cases of estimation errors, to find a robust plan based on an estimation interval with an acceptable performance in all the points present in that interval. This worth approach has been developed for centralized environment. However, in large scale environment imposes paying attention to the aspects of minimization of data transfer in the presence of low bandwidth and strong latency. In consequence, the placement of relational operators becomes critical.

Main efforts related to placement methods are to adapt the directed acyclic graph (DAG) scheduling approaches of parallel processing field [22] to parallel query processing. The methods proposed here are concentrated to apply tree decomposition with different methods and granularity [11, 36] in order to have schedulable components or to have different parallelization strategies: independent [7], pipelined [6, 13, 24, 25, 31, 35, 40] or partition parallelism [11, 15]. All these strategies are applicable to fast interconnected networks and homogeneous databases where the communication cost is low and sources are predictable. Nonetheless, none of the scheduling strategies developed in these fields meet the requirements of the large scale environment where the sources are heterogeneous and data and network characteristics are unpredictable. Simplified methods of scheduling applicable to large scale come from the field of distributed databases exploiting independent parallelism [10, 23, 26, 32, 41]. The main idea is to place the relational operators near to data sources or place them on the server site, again based on single-point estimations at compile-time.

6. Conclusion

In this paper, we presented a robust placement method to map, at compile-time, the mobile relational operators (MROs) of a query on execution sites over a large scale environment. MROs are self adaptive to changing run-time conditions. They make decision of migration in order to change their execution place if they discover estimation errors. Proposed placement methods, based on single-point estimation at compile-time, might lead bad performance in case of estimation errors.

We proposed: (i) to determine the migration space of a MRO which includes the sites on which the MRO is allowed to migrate during its execution, and (ii) to find a robust site which will allow acceptable response time for the estimation points in the estimation interval. The robust placement method makes the placement of MROs on execution sites based on an estimation interval. A site which allows acceptable performance over an estimation interval is chosen instead of a site giving the best performance on a single estimation point.

The experimental study focused on comparing the performance of mobile join operators (MJOs) of a query execution plan by single-point and robust placement methods in cases of estimation errors. For the robust placement method, we assumed to have an interval defined by the low, average (est) and high estimation points computed at compile-time related with the size of relations. Single-point placement method mapped the MJOs onto execution sites by finding the optimal performance based on average estimation, whereas the robust placement tried to place them by leaning on low, average and high estimations. We discovered that with a risk of loosing around 6% in response time, it is possible to gain up to 300% in some cases with the robust placement method.

7. References

[1] J.P. Arcangeli, et al. “Mobile agent based self-adaptive join for wide-area distributed query processing”. Journal of Database Management, 15(4), 2004, pp. 25-44.

[2] R. Avnur and J.M. Hellerstein. Eddies: “Continuously adaptive query processing”. Proc. of ACM SIGMOD, 2000, pp. 261-272.

[3] B. Babcock and S. Chaudhuri. “Towards a robust query optimizer”. A principled and practical approach. Proc. of ACM SIGMOD, 2005, pp. 119-130.

[4] S. Babu and P. Bizarro. “Adaptive query processing in the looking glass”. Proc. of 2nd Biennal Conf. on Innovative Data Systems Research (CIDR), 2005, pp. 238-249.

[5] S. Babu, et al. “Proactive re-optimization”. Proc. of ACM SIGMOD, 2005, pp. 107-118.

[6] C. Chekuri, et al. “Scheduling problems in parallel query optimization”. Proc. of ACM PODS, 1995, pp. 255-265.

[7] M.S. Chen, et al. “Scheduling and processor allocation for parallel execution of multi-join queries”. Proc. of ICDE, 1992, pp. 58-67.

[8] C. Collet and T.-T. Vu. “QBF: A Query broker framework for adaptable query evaluation”. Proc. of 6th Intl. Conf. on Flexible Query Answering Systems, 2004.

[9] W. Du, et al. “Query Optimization in a Heterogeneous DBMS”. Proc. of VLDB, 1992, pp.277-291.

[10] M.J. Franklin, et al. “Performance tradeoffs for client-server query processing”. Proc. of ACM SIGMOD, 1996, pp. 149-160.

[11] M.N. Garofalakis and Y.E. Ioannidis. “Multi-dimensional resource scheduling for parallel queries”, Proc. of ACM SIGMOD, 1996, pp. 365-376.

[12] D. Hagimont and L. Ismail. "Agents mobiles et client/serveur: evaluation de performance et comparison". Technique et Science Informatiques, 19(9). 2000, pp. 1223-1244.

[13] W. Hasan and R. Motwani. "Optimization algorithms for exploiting the parallelism-communication tradeoff in pipelined parallelism". Proc. of VLDB, 1994, pp. 36-47.

[14] M. Hussein, et al. "Embedded Cost Model in Mobile Agents for Large Scale Query Optimization", Intl. Symposium on Parallel and Distributed Computing, 2005, pp. 199-206.

[15] W. Hong and M. Stonebraker. "Optimization of parallel query execution plans in XPRS". In Proc. of the 1st Intl. Conf. on Parallel and Distributed Information Systems, 1991, pp. 218-225.

[16] A. Hulgeri and S. Sudarshan. "AniPQO: Almost non-intrusive parametric query optimization for nonlinear cost functions". Proc. of VLDB, 2003, pp. 766-777.

[17] Y. Ioannidis et al. "Parametric query optimization". In Proc. of VLDB, 1992, pp. 103-114

[18] Y. E. Ioannidis and S. Christodoulakis, "On the Propagation of Errors in the Size of Join Results", Proc. of ACM SIGMOD, 1991, pp. 268-277.

[19] Z. Ives, et al. "An adaptive data integration system for data integration". Proc. of ACM SIGMOD, 1999, pp. 299-310.

[20] Z. Ives, et al. "Adapting to source properties in processing data integration queries". Proc. of ACM SIGMOD, 2004, pp. 395-406.

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

[22] Y.K. Kwok and I. Ahmad. "Static scheduling algorithms for allocated directed task graphs to multiprocessors". ACM Computing Surveys, 31(4), 1999, pp. 406-471.

[23] L. Liu, et al. "Distributed query scheduling service: An architecture and its implementation". Intl. Jour. of Cooperative Information Systems, 7(2&3), 1998, pp. 123-166.

[24] B. Liu and E.A. Rundensteiner. "Revisiting pipelined parallelism in multi-join query processing". Proc. of VLDB, 2005, pp. 829-843.

[25] M-L. Lo, et al. "On optimal processor allocation to support pipelined hash joins". Proc. of ACM SIGMOD, 1993, pp. 69-78.

[26] L.F. Mackert and G.M. Lohman. "R* Optimizer validation and performance evaluation for distributed queries". Proc. of VLDB, 1986, pp 149-159.

[27] V. Markl, et al. "Robust query processing through progressive optimization". Proc. of ACM SIGMOD, 2004, pp. 659-670.

[28] F. Morvan and A. Hameurlain. "Dynamic memory allocation strategies for parallel query execution". Proc. of the 17th ACM Symposium on Applied Computing, 2002, pp. 897-901.

[29] F. Morvan, et al. "Mobile agent cooperation methods for large scale distributed dynamic query optimization". Proc. of DEXA Workshops, 2003, pp. 542-547.

[30] B. Özakar, et al. "Query optimization: mobile agents versus accuracy of the cost Estimation", Intl. Journal of Computer Systems Science and Engineering, 20(3), 2005, pp. 161-168.

[31] B. Özakar, et al. "Mobile join operators for restricted sources". Intl. Journal on Mobile Information Systems, 1(3), 2005, pp.167-184.

[32] M. Rodriquez-Martinez and N. Roussopoulos. "MOCHA: A self-extensible middleware system for distributed data sources". Proc. of ACM SIGMOD, 2000, pp. 213-224.

[33] D. Schneider and D.J. DeWitt. "A performance evaluation of four parallel join algorithms in a shared-nothing multiprocessor environment". Proc. of ACM SIGMOD, 1989, pp. 110-121.

[34] E.J. Shekita, et al. "Multi-join optimization for symmetric multiprocessors". Proc. of VLDB, 1993, pp. 479-492.

[35] A. Termehchi and M. Ghodsi. "Pipelined operator tree scheduling in heterogeneous environments". Jour. of Parallel and Distributed Computing, 63(6), 2003, pp. 630-637.

[36] K.L. Tan and H. Lu. "On resource scheduling of multi-join queries in parallel database systems", Information Processing Letters, 48(4), 1993, pp. 189-195.

[37] T. Urhan and M. Franklin. "XJoin: a reactively scheduled pipelined join operator". IEEE Data Engineering Bulletin, 23(2), 2000, pp. 27-33.

[38] S. Viglas. "Novel query optimization and evaluation techniques", PH. D. Thesis, Department of Computer Sciences, University of Wisconsin-Madison, 2003.

[39] G. Wiederhold. "Mediators in the architecture of future information systems", IEEE Computer, 25(3), 1992, pp. 38-49.

[40] J. Wu, et al. "Scheduling of query execution plans in symmetric multiprocessor database systems". Proc. of the 18th Intl. Parallel and Distributed Processing Symposium, 2004.

[41] Y. Zhou, et al. "An adaptable distributed query processing architecture". Data and Knowledge Engineering, 53(3), 2004, pp. 283-309.