Decentralized Dynamic Query Optimization based on Mobiles Agents for Large Scale Data Integration Systems

Mohammad Hussein[1, 2]
[1]Lebanese University, faculty of business, Tripoli, Lebanon
[2]Lebanese French University of Technology and Applied Sciences, Deddeh, El Koura, Lebanon

ABSTRACT
The query processing in large scale distributed mediations systems raises new problems and presents real challenges: efficiency of access, communication, confidentiality of access, availability of data, memory allocation. In this paper, we propose an execution model based on mobile agents for the distributed dynamic query optimization. In this model, each relational operator of an execution plan is executed by a mobile agent. Also, we embed into agent a migration policy allowing agent to choose an execution site among execution sites of the considered system. The performance evaluation shows that the proposed model improves the response time whatever the variation of estimations errors.

Keywords
Distributed mediation systems, Query optimization, Cost model, mobile agents.

1. INTRODUCTION
With the success and the fast evolution of network technology, the number of data sources accessible through these networks continues to grow. This growth has given rise to new applications referencing several data sources. Indeed, data on a topic is often scattered over several autonomous and heterogeneous sources. For example, consider the query that is to find the molecular functions of genes implicated in Alzheimer's disease. To produce the result of this query, it is first necessary to identify the genes of Alzheimer's disease by querying sources for genetic diseases such as GeneReviews (http://www.genetests.org/) and Genes and Disease (http:// www.ncbi.nlm.nih.gov/disease/). Then, the data sources describing molecular functions of genes (e.g. Gene Ontology, http:// www.geneontology.org/) are queried to determine the functionality of the identified genes.

In order to access uniformly on data from multiple sources, data integration systems based mediators and wrappers [5, 22, 28] were designed and developed. This architectural approach avoids copying all sources on a single site, which is not possible when the sources are distributed on a large scale. In this context, efficiently process queries present great difficulties. The optimizer's role is to generate, for a given SQL query, an optimal (or close to the optimal) execution plan from the considered search space. The optimization goal is to minimize response time and maximize throughput, while minimizing optimization costs. The general problem of the query optimization can be expressed as follows [14]: let a query q, a space of execution plans E, and a cost function cost, find the execution plan calculating q such as the cost(q) is minimum. Generally, an optimizer can be decomposed into three elements [14]: a search space corresponding to the virtual set of all possible execution plans, a search strategy generating an execution plan close to the optimal, and a cost model [28, 35] allowing to annotate operators' trees in the considered search space.

An execution plan generated by classical distributed query optimizer can have poor performance for four main reasons: (i) the centralized optimization, (ii) the imprecision of estimates, (iii) the unavailability of resources (i.e. data, CPU, network bandwidth, and memory). In response to these events, dynamic query optimization queries methods [1, 2, 9, 10, 21, 24, 25, 26, 29] have been developed. These methods correct, at run-time, the sub-optimality of execution plans. The majority of proposed dynamic query optimization methods are centralized. This centralization creates a bottleneck and produces a relatively large exchange messages on a low speed network and high latency. Thus, it becomes important to make autonomous and self-adaptable query execution.

In this paper, we present an approach proposes autonomous and mobile execution of every relational operator of an execution plan, able to react in an autonomous and decentralized way with the evolution of the system state (e.g. workload of sites, bandwidth) and with the estimation errors. Furthermore, it can move from one site to another to continue its execution. The site of migration of the agent is chosen, among a set of sites determined by the optimizer, according to several metrics: the workload of sites, the data unavailability, the costs to produce the data, the cost to send the result and the cost of the agent migration.

The remainder of paper is organized in the following way: in the section 2 we describe state of the art of the main optimization dynamic methods. Section 3 proposes an execution model based on mobile agents for dynamic query optimization. Section 4 presents the experimentation environment and the experiments results. Finally, we conclude and present the perspectives.

2. RELATED WORKS
In data base systems, the optimization methods can be classified in two types [16]: static and dynamic. A dynamic optimization method is characterized by three elements [18] : (i) it receives information from its environment, (ii) uses this information to determine its behavior, and (iii) it carries out the steps (i and ii) iteratively, which enables a dynamic optimization method to adapt the drawings to changes in the execution environment.

The proposed dynamic optimization methods are distinguished according to several aspects. For example, they correct sub-optimality of the execution plans in various manners. Moreover, the correction level of the execution plans can be also different from a method to another. These methods can be addressed for various environments (single processor, parallel and distributed) and to adapt to different events. The table 1 presents a set of parameters allows differentiating the methods. Some of these parameters were quoted or described in several reviews [17, 18, 15, 30].

Table 1: comparison of dynamic optimization methods

Methods	*Decision making*	*Actions correcting the sub-optimality of execution plans*	*Causes leading to the sub-optimality of execution plans*	*Modification time of the sub-optimality of execution plans*
[3]	*centralized*	*re-scheduling*	*all*	*intra-operator*
[34]	*centralized*	*re-scheduling*	*all*	*intra-operator*
[12]	*centralized*	*re-scheduling*	*estimation errors*	*inter-operator*
[24]	*centralized*	*re-optimization*	*estimation errors*	*inter-operator*
[5]	*centralized*	*re-optimization*	*delays* *memory unavailable*	*inter-operator*
[1,2]	*centralized*	*re-optimization*	*delays*	*inter-operator*
[21,22]	*centralized/ decentralized*	*re-optimization* *dynamic operator*	*all*	*inter-operator/ intra-operator*
[20]	*decentralized*	*Dynamic operator*	*Delays* *user preferences*	*intra-operator*
[19,27]	*decentralized*	*re-optimization*	*all*	*intra-operator*
[9]	*decentralized*	*re-optimization*	*all*	*inter-operator*
[23]	*decentralized*	*re-optimization*	*estimation errors*	*inter-operator*

2.1 Decision making

A characteristic of dynamic optimization methods is the decision making nature of the modifications of the execution plans (*decision making* column in the Table 1). We distinguish two classes: centralized methods [1, 2, 3, 4, 5, 12, 24] and decentralized dynamic optimization methods [9, 19,20, 21, 23,27]

A method of optimization known as centralized if it is based on a main process which is charged to supervise, control and modify the execution plans. This process can be based on other modules ensuring the production of necessary information for the control and the modifications of an execution plan. However, centralization does not allow these methods to scale due to: (i) the relatively large number of messages on a low speed network and high latency, and (ii) the bottleneck of the optimizer because all messages converge to a single point.

Decentralization of control and changes in execution plans in dynamic optimization methods avoid the bottleneck of centralization. For a flexible, dynamic and decentralized execution of the queries submitted to the large scale distributed mediation systems, [9] proposed an execution model based on broker. The broker is the basic unit of the execution of a query. The process of execution of a query can be compared with a set of brokers. Each broker ensures the execution of a sub-query. Thus, an execution plan of a query is represented by a tree which nodes are the brokers who are charged to execute the sub-queries. The broker supervises the execution of his sub-query, it detects the inaccuracies and it adapts it according to these inaccuracies. Moreover, it can communicate with the user and the other brokers to take into account the changes of the execution environment. In the methods proposed by [21], the join operators are autonomous, decentralized and dynamic in order to deal with the changes in the execution environment. This method improves the cost of local processing by adapting the use of CPU, I/O and memory to changes in the execution environment (e.g. estimation error, delays of data arrivals rates). However, the proposed methods focus mainly on the resources (CPU, I / O and memory) and do not consider network resource. In this method, the volume of data transferred determined at compile-time remain the same regardless of the estimation errors observed during execution.

2.2 Actions correcting the sub-optimality of execution plans

The actions used for correcting the sub-optimal execution plans can be classified in four types : (i) replacement,(ii)

scheduling, (iii): re-optimization, and (iv) uses of dynamic operators. A solution to correct the sub-optimal execution plans consists of changing the operations sequences of the execution plan at runtime [31] proposes strategies of dynamic re-scheduling of the operators inter-sites (e.g. joint, union) in order to react to the inaccuracies of estimation. The operators inter-sites can be executed as soon as two sub-queries executed on different sites produced their results. These strategies are based on the real execution times of the sub-queries instead of their estimated costs. They use the partial results available at runtime to define the scheduling of the executions between the operator inter-sites. In this method, processing of the queries is done in two steps [12]: (i) Compilation: during this stage, a global query is decomposed into local sub-queries. The sub-queries are sent in parallel to the sources in order to be executed. (ii) Dynamic scheduling: this step defines a dynamic scheduling between the operations consuming of the results of sub-queries sent on sites. As soon as a sub-query produces its result, a threshold is associated with the result. This threshold is used to determine whether the result must be consumed immediately to execute a join with another available result, or the consumption of this result will be delayed while waiting for another one which is unavailable in this moment. The threshold associated with a result is calculated according to the costs and of the selectivity factors of all join connected to this result. Thus, the executions of the joins costs are higher than the thresholds associated with their operands which are delayed, and the join of lower and most selective cost is executed.

The alternative approach, with the operators re-scheduling, is the re-optimization [1, 4, 5, 21]of the execution plans. This approach can produce for the remainder of the query a sub-plan completely different from that generated during compilation. It can also introduce new operators or change scheduling between the operators. Several methods were proposed in the context of this approach. Each one of them was addressed to a specific environment and was designed to deal with the sub-optimality caused by one or more events (e.g. errors in estimation). For example, the technique of query scrambling [1,2] proposes solutions to deal with the data arrival rates. Another method [24] proposes solutions to react to the estimation errors.

2.3 Causes leading to the sub-optimality of execution plans

Each dynamic optimization method is able in adapting the execution plans in order to react to one or several causes. Methods have been proposed to react to each of the following causes:

(i) **Estimation errors**: in several cases, the statistics describing the data are unavailable or inaccurate. A solution to solve this problem is to collect statistics at runtime. The methods which are based on this principle re-optimize the remainder of the execution plan as soon as these statistics become available.

(ii) **Memory available**: the memory allocated to the operators of an execution plan can become available because of the simultaneous and concurrent executions of several queries or any other external event. For that, several methods try to adapt to the memory unavailability at runtime.

(iii) **Delays in data arrival rates**: in a large scale environment, the access to the data implies a great number of distant data sources, intermediate sites and links of communication which are vulnerable to the congestion. The congestion can generate unexpected delays in data arrivals. Thus, methods propose some solutions to adapt to this event.

(iv) **Users Preferences:** the users have preferences to receive certain partial results as soon as possible. Thus, the users can classify the tuples of results according to their degree of importance. In this case, methods adapt their behaviors in order to accelerate the production of the most important tuples of result.

2.4 Modification time of the sub-optimality of execution plans

The modification level of the executions plans can also be differing from a method to another. Methods propose to modify the execution plans either between executions of two operators, or after materialization of the temporary relations. What we will call modification on the *inter-operator level* in the remainder of the paper. Others propose to modify the execution plans during the execution of a physical operator. What we will call modification on the *intra-operator level* in the remainder of the paper. Sometimes two levels of modifications are combined in only one method.

To modify the sub-optimal execution plans on the intra-operator level, two approaches were proposed: the first is based on the routing of tuples named *Eddy* [3], and the second is based on the dynamic partitioning of data [22]. *Eddy* [3] is a mechanism of query processing which changes continuously the execution schedule of operators in order to adapt to the changes of the execution environment. *Eddy* can be considered as a router of tuples positioned between a number of data sources and a set of operators. Each operator participate in an *Eddy* must have one or two input queues to receive the tuples sent by *Eddy* and an output queue to return the results tuples to *Eddy*. The tuples received by an Eddy are redirected towards the operators in different orders. Thus, the scheduling of the operators is encapsulated by the dynamic routing of tuples. The method proposed by [22] corrects the sub-optimality of execution plans relied on the dynamic data partitioning. The execution plan of a query is constantly supervised, at runtime, and it can be replaced by a new plan in the case where we consider that the current plan is sub-optimal. The tuples processed by each used plan represent a data partitioning which is dynamically given. When an execution plan is replaced, a new data partitioning is produced. The dynamic optimization methods described in [3, 22] carry out the modification of the sub-optimal execution plans during the execution of an operator (intra-operator). Other approaches propose to modify the sub-optimal execution plans after the materialization of a temporary relation or after the termination of the execution of an operator. These modifications vary from operators re-scheduling of execution plan, re-optimization of the remainder of the plan until replacement of the execution plan.

3. EXECUTION MODEL BASED MOBILE AGENTS FOR DECENTRALIZED DYNAMIC QUERY OPTIMIZATION

In this section, we describe an execution model based on mobile agents for decentralized dynamic queries optimization. During the query optimization, the optimizer of mediator generates an execution plan including : (i) the number of mobile agents participating in the evaluation of execution

plan, (ii) the operator will be executed by each agent, and (iii) the execution site of each agent. In the rest of this section, we present the mobile join algorithms [20]. Then, we propose an execution model based on mobile agents for queries including cooperation policy allowing the agents to run efficiently the execution plans. Finally, we describe a migration policy of mobiles agents based on two cost models: the mediator cost model and the embedded cost model in a mobile agent.

3.1 Mobile join algorithms

In a distributed environment an interesting aspect in the query optimization is the selection of execution sites of the operators. The unary operators (e.g. selection, projection) are placed on the sites of their operands. However, for the binary operators (e.g. join, union), the optimizer chooses a site in order to execute these operators. Another interesting aspect is the execution of joining two operands residing on different sites. In literature, there are two approaches to execute the joining of two operands residing on different sites: (i) the direct join, and (ii) the join based semi-join [7].

In order to deal with the unexpected changes in large scale distributed environment, we propose to use the mobile agent [16, 25] for extension of join algorithms which is called mobile join for two reasons: (i) they are executed by mobile agents, and (ii) they can change their execution site locations. This extension allows the join to change their execution site. The decision and change control of the execution site are made in a decentralized and autonomous manner. It is no longer the optimizer chooses the join execution site, but the join itself chooses its execution site. Indeed, the mobile agent executing a join adapts to changes in characteristics of the execution environment (e.g. network bandwidth, available memory, estimations errors). To simplify the rest of this article we present briefly the simple mobile hash join algorithm. In the simple hash join algorithm, during the building of the hash table, the characteristics of R1 (e.g. size, values distribution of every attribute) can be calculated precisely. So, from the precise statistics of R1, the statistics of R2 estimated at the optimization, the statistics revised of T from R1 and R2, the unavailability of R2 and the state of the system, it is possible to make a decision on the localization of the probe step and eventually move this step to another site.

3.2 Mobile query

In this sub-section, we propose an execution model based on mobile agents for queries in distributed mediations systems. In this model, each operator execution plan is executed through a mobile agent. During the generation of execution plan, the agents are associated with the operators of the execution plan and also the initial execution site locations are determined. These agents can be placed either on the mediators or the wrappers. Agents placed on mediators can migrate at run-time from site to another. Instead, the agents placed on wrappers are static and they finish their executions on the wrappers selected during the optimization step. The association between operators and mobile agents is illustrated in Figure 1.

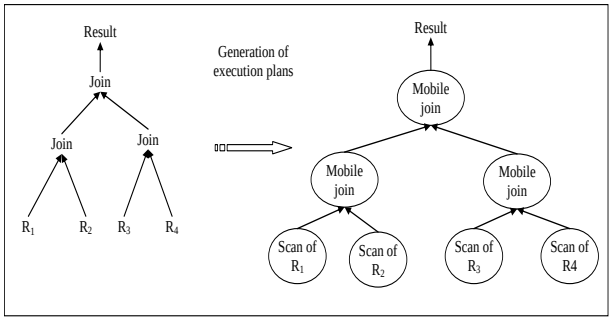

Fig. 1 : mobile agents associated with an execution plan

3.2.1 Cooperation policy

We recall that a mobile agent executing a mobile join can change its execution site in order to improve its response time. Thus, this agent can migrate its current execution site to the new chosen one. This migration from one site to another is taken autonomously and decentralized. The decision to migrate for an agent cannot be made without taking into account the decisions of the other communicating agents. Indeed, a global optimum cannot be reached by a sum of local optimum. For example (Figure 2), let us consider two joins J1: T = Join(R1, R2) and J2: res=Join(T, R3) processing respectively on the sites S1 and S2. Now, if J1 decides to migrate on S2 in order to minimize the communications between S1 and S2. Also J2 decides to migrate on S1 for the same reasons. We will have, thus, a lack of cooperation between J1 and J2. Another example of cooperation concerns the propagation of the estimation corrections. Indeed, the join operator J1, establishing of an error on R1 after building the hash table, can propagate the error correction to J2. This propagation will lead, eventually, J2 to migrate on another site. In the rest of the paper, we will name the various communicating agents with a join J noted AJ, in the following way:

– AR1 and AR2 are the agents producing R1 and R2,
– AT is the agent consuming the join result.

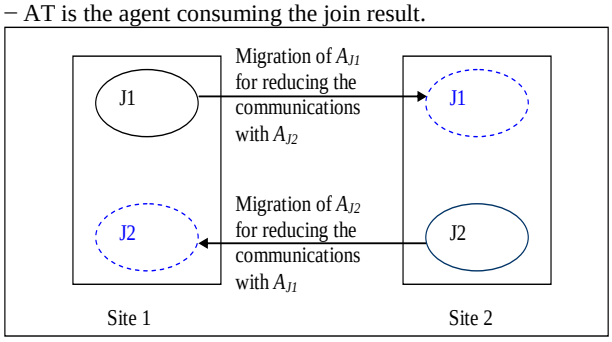

Fig. 2 : Example of not cooperation between two agents

We have seen above, the need to make a cooperation policies between agents involved in the evaluation of a query. The agent needs to make a decision about his migration are three fold: (i) the execution site about other agents communicating with him, (ii) the estimate data volume, and (iii) the resources availability (e.g. CPU, network bandwidth). In this sub-section, we focus on cooperation between agents from the same query. Thus we focus on the knowledge of the execution sites of agents, on the estimation of the data volume and on the data unavailability.

In order to design an efficient cooperation methods [27] and to study the impact of the number of exchanged messages between the agents. The proposed cooperation policy [27]

consists in the cooperation of the agents to know the execution sites of the agents which communicate between them. For an agent AJ processing a hash join (R1, R2), the decision to migrate is made after the Build step and before the Probe step. Thus, after its decision, the only communicating agents with it will be AR2 and AT. To minimize the communications between the agents during its decision phase, this one is taken hierarchically. The agent AJ will make its decision to migrate after AT made its decision. Thus, when AJ makes its decision, it knows the site of AT and the site of AR2 since AR2 cannot migrate as long as AJ did not make its decision. The advantage of this method is that it also avoids the duplication of the data messages. Indeed, AJ knows the AT execution site. Thus AJ can sent its data directly to AT without passing by an intermediary. We do not need to set up strategies such as the forwarding pointer or the use of a central server which duplicate part of the data messages.

Another factor strongly influencing the response is the precision of the data volume estimations. In addition to the communication of the execution sites of the agents, we propagate the correction of the estimation errors. After the hash table building, the R1 characteristics are known more precisely. If AR2 is able to improve the R2 estimation and to communicate this result to AJ, the estimation of the data volume processed by AJ could be only more precise. Moreover, AJ can communicate to AT the new estimated characteristics of T. Hence, the decisions made by AJ and AT will be better. The propagation of estimation corrections is processed in the opposite direction of the site communication.

3.2.2 Migration policy

Let us remind that a mobile agent can move from a mediator to another. The migration site is chosen by the mobile agent according to several metrics. This metrics are computed by the mediator where the mobile agent builds its hash table. The problem here is that this mediator does not have all necessary information to compute the metrics of this mobile agent (e.g. operands profile, the production costs of operands). In order to assure the autonomy of the mobile agent, this one needs to provide for the mediator whe : (i) the profile of the second operand (R2), (ii) The sites where it can move, and (iii) the parameters involved in the formula estimating the production costs of the second operand (R2). These components are necessary to compute the metrics of the agent and they constitute the agent cost model. In this section we describe the various parts of the cost model intervening in the estimation of the cost of the execution plans. We distinguish between two cost models involved in the estimation of executions plans costs: the part residing in the mediator which is called mediator [28] cost model and the part embedded in the agent which is called the agent cost model [19]

4. PERFORMANCE EVALUATION

This section compares the results obtained with the execution model based on mobile agents and with a classical execution. We study the behavior of proposed model and the classical according to the tree structure of the execution plan generated by an optimizer. In the following sub-sections, we describe the experimentation environment, the query experimentation and the experiments.

4.1 Experimentation environment

We realized our experiments in distributed environment. It is constituted of five workstations (HP) interconnected by Internet. The all workstations are located in Lebanon at different cities. These workstations will be called respectively Tripoli, Beirut, Saida, Baalbeck, and Zahle. To handle our experiments, we installed on every workstation a platform of mobile agents [13] including the mobile hash join algorithm, the cooperation policy and the migration policy. In this one, every mobile agent runs on a Virtual Machine Java (JDK 1.6.2). The response times are measured by real executions. These are carried out between sites interconnected via a network. Our experiments are handled in multi-user environments, where several users can start up applications. In these environments, it is difficult to reproduce an experiment in identical conditions. Indeed, the workload of an execution site varies from moments to another (available memory, number of running processes, etc.) and the amount of the data transferred through the network also varies.

The costs associated with the execution of these algorithms are deduced by calibration. The migration cost of an agent given in the table Tab.2 includes the serialization cost, the transfer cost and the de-serialization cost. Of course, when the agent migrates with its data, it must be added the serialization cost, the transfer and the de-serialization of the data which is proportional to its size.

Table 2: environment parameters

Networks parameters		
	Banwidth (KB/s)	Time to send a page(ms)
[Tripoli, Beirut, Saida, Baalbeck, Zahle ←→ Tripoli, Beirut, Saida, Baalbeck, Zahle]	[104.3-112.7]	[727.89-749.52]
Mobile agent parameters		
		Agent migration(ms)
[Tripoli, Beirut, Saida, Baalbeck, Zahle ←→ Tripoli, Beirut, Saida, Baalbeck, Zahle]		[28457- 29713]
Workstations parameters		
	Time to write a page (ms)	Time to read a page (ms)
[Tripoli, Beirut, Saida, Baalbeck, Zahle]	[0.71-0.79]	[0.58-0.65]

4.2 Experimentation query

To compare quantitatively the various methods, we consider that four relations R1, R2, R3 and R4 are respectively on the sites: Tripoli, Beirut, Saida and Baalbeck. The relation size estimated by the optimizer are respectively 15 000, 30 000, 30 000 and 37 500 tuples. The join selectivity factor Ri ∞ Rj is 1.5/Max (||Ri||, ||Rj||) where ||Ri|| indicates the Ri number of tuples [6, 8]. The query Q = R1 ∞ R2 ∞ R3 ∞ R4 is expressed on a Zahle site. This query is constituted of tree joins called J1, J2 and J3. Every join is executed by a mobile agent. The migration space of the cost model embedded in each agent is

constituted of the all sites Tripoli, Beirut, Saida, Baalbeck, and Zahle.

4.3 Experimentation results

In this sub-section, we describe the results of the experiments handled for the execution model based on mobile agents that we will call mobile execution and the traditional method that we will call classical execution applied to a left deep tree, right deep tree and bushy tree. Here, we present and discuses the behavior of the proposed model according to the variation between the parameters estimated at compile-time (the estimated number of tuples of R1 noted ||R1es||) and that computed by agent at runtime (the computed number of tuples of R1 noted ||R1comp).

4.3.1 Results applied to left deep tree

In the optimal execution plan left deep tree of the query q, the joins J1 : T1= R1 ∞ R2, J2 : T2= T1 ∞ R3 and J3 : Res = T2∞ R4 are respectively on the site Tripoli, Beirut and Zahle.

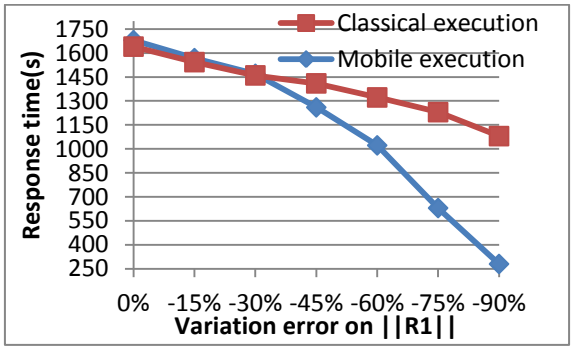

Fig. 3 :Left deep tree performance by decreasing ||R1||

Figure 3 shows the response time of mobile execution and the classical execution by decreasing the R1 number of tuples. We observe that the response times of mobile execution are slightly higher than classical execution (about 4%) when the error on ||R1|| is between 0% and 30%. This difference is mainly due to the mobility overhead. This overhead takes into account the agent serialization, the agent migration and the agent start-up on its new site. From -30%, mobile execution obtain better performance (between 10% and 70%) by modifying the execution plan generated by the optimizer. We obtain the best performance by moving joins 2 and 3 before their execution.

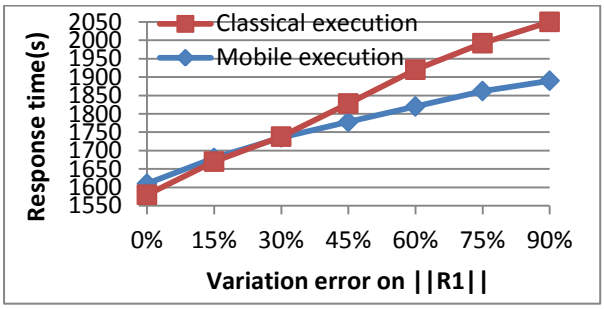

Fig. 4 :Left deep tree performance by increasing ||R1||

The curves of Figure 4 shows the response time of each method by increasing the R1 number of tuples. We note that the performance of mobile execution is even worse than those of classical execution (between 3% and 4%) when the error on ||R1|| is between 0% and 30%. On the other hand, when the error is higher than 30%, mobile execution obtains better performance (between 3% and 9%) by moving j2 on Zahle and by moving the Probe of the j1 on Zahle.

4.3.2 Results applied to right deep tree

In the optimal execution plan left deep tree of the query q, the joins J1 : T1= R1 ∞ R2, J2 : T2= R3 ∞ T1 and J3 : Res = R4 ∞ T2 are respectively on the site Tripoli, Saida and Baalbeck.

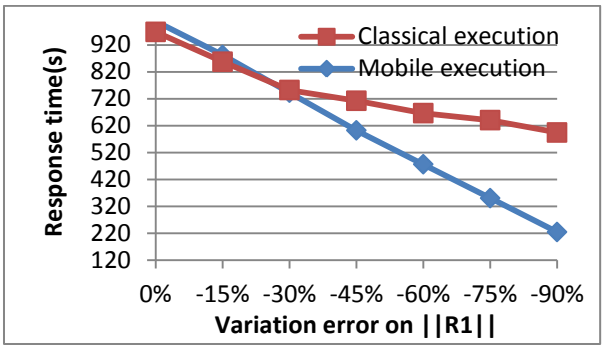

Fig. 5 : Right deep tree performance by decreasing ||R1||

Figure 4 shows the response time of mobile execution and the classical execution by decreasing the R1 number of tuples. We observe that the response times of mobile execution have a slightly higher response time between 0% and -20%. From -30%, they improve the performance compared to standard (between 10% and 75%). same behavior. Indeed, on this tree, mobile execution cannot anticipate the migration of a join since all the Build steps of the joins are started at the same time.

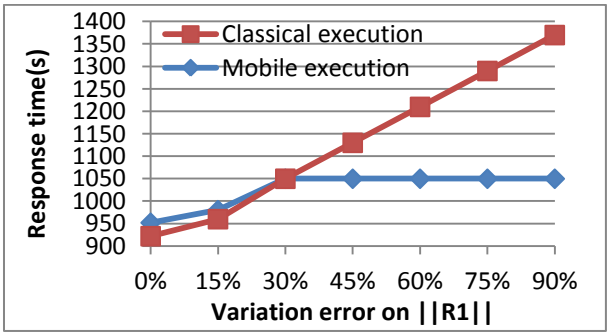

Fig. 6 : Right deep tree performance by increasing ||R1||

The curves of Figure 6 shows the response time of each method by increasing the R1 number of tuples. We notice that between 0% and 20% , mobile execution have a response time slightly higher than standard approximately 4%. From 30 % mobile executions have a better response time by moving all the join on Zahle after the Build step

4.3.3 Results applied to bushy deep tree

In the optimal execution plan left deep tree of the query q, the joins J1 : T1= R1 ∞ R2, J2 : T2= R3 ∞ R4 and J3 : Res = T1∞ T2 are respectively on the site Tripoli, Beirut and Zahle.

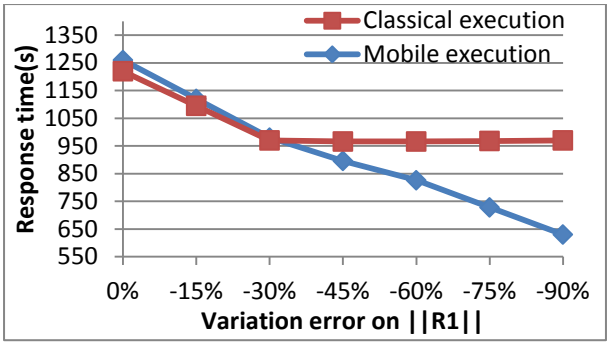

Fig. 7: Bushy deep tree performance by decreasing ||R1||

The curves of Figure 7 show response time of each method by decreasing the number of tuples of R1. We observe that between 0% and -30%, mobile execution have a higher response time (approximately 4%) compared to classical execution. In this bushy deep tree, an estimation error on ||R1|| has impact only on J1 and J3. Moreover J2 (i.e. R3 ∞ R4) is the most significant join of this plan. In this case, the migration of the J1 and J3 following an error on R1 have little impact on the performance of the bushy execution. Thus, the performance improvements of mobile execution are relatively weak.

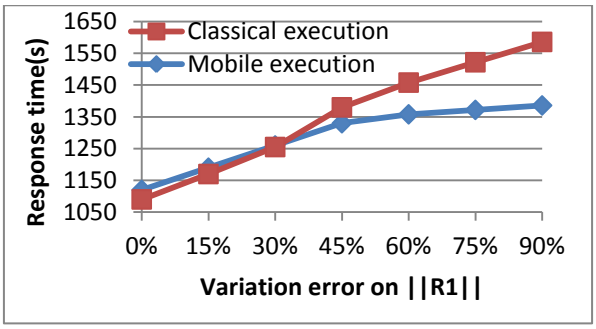

Fig. 8: Bushy deep tree performance by increasing ||R1||

We observe the same behavior between mobile execution and the classical execution when the number of tuples of R1 increases (Figure 8). From 90% of error on R1, the J1 migration starts to have really an impact over the response time because the join J1 is close the data volume processed by the join J2.

5. Conclusions and perspectives

In a large-scale mediation system, centralized dynamic optimization methods cannot be valid due to the centralization of the control and the modifications of execution plans. In this paper, we proposed an execution model based on mobile agents and cost model approaches for large scale distributed query optimization. Also, we show mobile agents who can react dynamically to the estimation errors of the cost models and to the data unavailability. In this context, to guarantee the autonomy of an agent we propose a cooperation and migration policies for the mobile agents executing a relational operator. The proposed cooperation policy consists in the cooperation of the agents to know the execution sites of the agents which communicate between them and the propagation of correction estimations errors. The migration policy of an agent is based on embedded cost model in the agent which must be as small as possible in order to minimize the agent migration cost on a large scale network.

In the next future we will develop a decision policy considering, in an incremental way, the data availability, the workload of the sites, memory allocation on the sites, as well as the network bandwidth heterogeneity. This new policy will have to minimize to the maximum the number of messages exchanged between mobiles agents. Also, we plan to define methods which determine the migration space of the agents participating in execution of a query. These methods must take into account the migration spaces of the other agents participating in the execution of a query and the tree structure of execution plan of the query. Finally we will extend our performance evaluation in order to study the behaviors of the agents on the level of the complex queries.

6. REFERENCES

[1] L. AMSALEG et al.; Scrambling query plans to cope with unexpected delays, Proc. of the Fourth International Conference on Parallel and Distributed Information Systems, IEEE Computer Society, Miami, Florida, USA, December 1996, pp. 208-219.

[2] L. AMSALEG, M. FRANKLIN, A. TOMASIC; Dynamic query operator scheduling for wide-area remote access, Distributed and Parallel Databases, vol. 6, no3, Kluwer Academic Publishers, 1998, pp. 217-246.

[3] R. AVNUR, J.-M HELLERSTEIN; Eddies: continuously adaptive query processing, Proc. of the ACM SIGMOD International Conference on Management of Data, ACM Press, Dallas, Texas, USA, May 2000, pp. 261-272.

[4] S. Babu, P. Bizarro, D. -J. DeWitt; Proactive Re-optimization. Proc. of the ACM SIGMOD International Conference on Management of Data, ACM Press, Baltimore, Maryland, USA, June 2005, pp.107-118.

[5] L. BOUGANIM and al.; A dynamic query process-ing architecture for data integration systems. Journal of IEEE Data Engineering Bulletin, IEEE Computer Society, vol. 23, no2, June 2000, pp. 42-48.

[6] N. BRUNO, S. CHAUDHURI; Efficient Creation of Statistics over Query Expressions, Proc. of the 19th International Conference on Data Engineering, IEEE Computer Society, Bangalore, India, March 2003, pp.201-212.

[7] D.-M. Chiu, Y.-C. Ho ; A Methodology for Interpreting Tree Queries Into Optimal Semi-Join Expressions, Proc. of the ACM SIGMOD International Conference on Management of Data, ACM Press, Santa Monica, California, USA, Mai 1980, pp. 169-178.

[8] C.-M. CHEN, N. ROUSSOPOULOS; Adaptive Selectivity Estimation Using Query Feedback, Proc. of the ACM SIGMOD International Conference on Management of Data, ACM Press, Minneapolis, Minnesota, USA, May 1994, pp. 161-172.

[9] C. COLLET, T.-T. VU ; QBF: A Query Broker Framework for Adaptable Query Evaluation, Proc. of 6th International Conference on Flexible Query Answering Systems, Springer Verlag Publishers, Lyon, France, June 2004, pp. 362-375.

[10] A. DESHPANDE, J.-M. HELLERSTEIN; Lifting the Burden of History from Adaptive Query Processing, Proc. of the Thirtieth International Conference on Very Large Data Bases, Morgan Kaufmann, Toronto, Canada, August 2004, pp. 948-959.

[11] R.-S. EPSTEIN, M. STONEBRAKER, E. WONG ; Distributed Query Processing in a Relational Data Base System, Proc. of the ACM SIGMOD International Conference on Management of Data, ACM Press, Austin, Texas, June 1978, pp. 169-180.

[12] C. EVRENDILEK et al.; Multidatabase Query Optimization, Journal of Distributed and Parallel Databases, Kluwer Academic Publishers, vol 5, no1, January 1997, pp. 77-113.

[13] A. FUGGETTA, G.-P. PICCO, G. VIGNA; Understanding Code Mobility, IEEE Transactions on Software Engineering, IEEE Computer Society, vol. 24, no5, May 1998, pp. 342-361.

[14] Ganguly, S., Hasan, W., Krishnamurthy, R.: Query Optimization for Parallel Execution. In: Proc. of the 1992 ACM SIGMOD, vol. 21, pp. 9–18. ACM Press, San Diego (1992)

[15] A. GOUNARIS and al.: Adaptive Query Processing: A Survey, Proc. of the 19th British National Conference on Databases, Sheffield, UK, July 2002, pp. 11-25.

[16] A. GOUNARIS and al.: Resource Scheduling for Parallel Query Processing on Computational Grids. In: Proc. of the 5th IEEE/ACM Intl. Workshop on Grid Computing, pp. 396–401 (2004).

[17] A. HAMEURLAIN, F. MORVAN; Parallel Query Optimization Methods and Approaches: a Survey, Journal of Computers Systems Science & Engineering, CRL Publishing Ltd9 De Montfort Mews, vol. 19, no5, September 2004, pp. 95-114.

[18] J.-M. HELLERSTEIN et al.; Adaptive query processing: Technology in evolution, IEEE Data Engineering Bulletin, IEEE Computer Society, vol. 23, no2, June 2000, pp. 7-18.

[19] M. Hussein, F. Morvan, A. Hameurlain ; Embedded Cost Model in Mobile Agents for Large Scale Query Optimization, Proc. of the 4th International Symposium on Parallel and Distributed Computing, IEEE Computer Society, Lille, France, Juillet 2005, pp. 199-206.

[20] M. Hussein, Mobile Join Algorithms based on Mobiles Agents for Large Scale Distributed Query Optimization. International Journal of Applied Information Systems (IJAIS), Volume 4– No.1, September 2012 , pp 55-61.

[21] Z.-G. IVES et al.; An Adaptive Query Execution System for Data Integration, Proc. of the ACM SIGMOD International Conference on Management of Data, ACM Press, Philadelphia, Pennsylvania, USA, June 1999, pp. 299-310.

[22] Z.-G. IVES, A.-Y. HALEVY, D.-S. WELD; Adapting to Source Properties in Processing Data Integration Queries, Proc. of the ACM SIGMOD International Conference on Management of Data, ACM Press, Paris, France, June 2004, pp. 395-406.

[23] R. JONES, J. BROWN; Distributed Query Processing Via Mobile Agents, find the 14 november 2002, accessible via: http://www.cs.umd.edu/~rjones/paper.html, 1997.

[24] N. KABRA, D.-J. DEWITT; Efficient Mid-Query Re-Optimization of sub-optimal query execution plans, Proc. of the ACM SIGMOD International Conference on Management of Data, ACM Press, Seattle, Washington, USA, June 1998, pp. 106-117.

[25] L. KHAN, D. MCLEOD, C. SHAHABI; An Adaptive Probe-Based Technique to Optimize Join Queries in Distributed Internet Databases, Journal of Database Management Idea Group, vol. 12, no4, Octobre 2001, pp. 3-14.

[26] F. MORVAN, A. HAMEURLAIN; Dynamic Memory Allocation Strategies For Parallel Query Execution, Proc. of the ACM Symposium on Applied Computing, ACM Press, Madrid, Spain, March 2002, pp. 897-901.

[27] F. MORVAN, M. HUSSEIN, A. HAMEURLAIN ; *Mobile Agent Cooperation Methods for Large Scale Distributed Dynamic Query Optimization*, Proc. of the 14th International Workshop on Database and Expert Systems Applications, IEEE Computer Society, Prague, Czech Republic, Septembre 2003, pp. 542-547.

[28] H. NAACKE, G. GARDARIN, A. TOMASIC ; Leveraging Mediator Cost Models with Heterogeneous Data Sources, Proc. of the Fourteenth International Conference on Data Engineering, IEEE Computer Society, Orlando, Florida, USA, February 1998, pp. 351-360.

[29] B. NAG, D.-J. DEWITT; Memory Allocation Strategies for Complex Decision Support Queries, Proc. of the ACM CIKM International Conference on Information and Knowledge Management, ACM Press, Bethesda, Maryland, USA, November 1998, pp. 116-123.

[30] M. OUZZANI, A. BOUGUETTAYA; Query Processing and Optimization on the Web, Distributed and Parallel Databases, Kluwer Academic Publishers, vol. 15, no3, May 2004, pp. 187-218.

[31] F. OZCAN et al. ; *Dynamic query optimization in multidatabases*, Data Engineering Bulletin, IEEE Computer Society, vol. 20, n°3, Septembre 1997, pp. 38-45.

[32] V. RAMAN, A. DESHPANDE, J.-M. HELLERSTEIN; Using State Modules for Adaptive Query Processing, Proc. of the 19th International Conference on Data Engineering, IEEE Computer Society, Bangalore, India, March 2003, pp. 353-362.

[33] G.-M. SACCO, S.-B. YAO; Query Optimization in Distributed Data Base Systems, Advances in Computers, vol. 21, 1982, pp. 225-273.

[34] Y. ZHOU et al. ; An adaptable distributed query processing architecture, Data & Knowledge Engineering, vol. 53, no3, June 2005, pp. 283-309.

[35] Q. ZHU, S. MOTHERAMGARI, Y. SUN; Cost Estimation for Queries Experiencing Multiple Contention States in Dynamic Multidatabase Environments, Journal of Knowledge and Information Systems, Springer Verlag Publishers, vol. 5, no1, Februray2003, pp. 26-49.