Query Brokers for Distributed and Flexible Query Evaluation

Tuyet-Trinh Vu Christine Collet

***Abstract*— This paper presents our work on supporting flexible query evaluation over large distributed, heterogeneous, and autonomous sources. Flexibility means that the query evaluation process can be configured according to application context-specific, resources constraints and also can interact with its execution environment.**

Our query evaluation is based on *Query Brokers* as basic units which allow the query processing to interact with its environment. Queries are evaluated under query brokers contexts defining constraints of the evaluation task. Query Brokers have dynamic behaviors towards their execution environment. This paper focuses on the definition and the role of query brokers in the query evaluation task in large-scale mediation systems. We show also how query brokers ensure the flexibility of this task.

I. Introduction

The increasing use of computers and the development of communication infrastructures have led to a wide range of information sources available to access through networks. This gives the possibility of sharing information among different user groups and applications.

Many approaches such as data warehousing, mediation, search engine, etc. have been proposed to give an uniform access to multiple data sources [5]. Among them, the mediation approach proposed initially in [27] for simplifying, abstracting, reducing, merging and explaining data seems still be the most suitable for querying a large number of data sources which are structured, semi-structured and often changed. Generally speaking, in mediation systems queries are formulated over a global schema, called also the mediation schema. These queries, called global queries, are then rewritten on local schemas, i.e. schemas of the component sources, and decomposed into sub-queries, called local queries. The sub-queries are evaluated by their appropriated sources and intermediate results are assembled by mediators.

Many works in data integration such as [17], [7], [4], [19], [16], [13] follow this reference mediation architecture. However, these works aim at solutions for different problems of data integration so they have different mediation architectures in terms of their data model, the level of distribution and autonomy of their sources systems. The focuses of works in [22], [7] are on developing models for integrating heterogeneous sources. The emphasis of work in [10] are generating wrappers or translators. The authors of [17], [20] focus on problems of reformulating global queries into local queries. The works in [16] stress on modeling restricted capabilities of data sources as tables with binding patterns and optimizing queries in the presence of binding patterns. Nowaday, the augmentation of the number of sources makes new challenges to integrate and query data in large distributed mediation systems. As data sources are largely distributed over network, the autonomy of sources and the unpredictability of execution environment are more critical.

Our focus is on querying data in large-scale distributed mediation systems. We take into consideration the distribution and the autonomy of sources. Such autonomous sources can communicate their meta-information to mediators as they want. Consequently, the knowledge of sources collected by mediators could not be complete or not be up to date. Besides, it may be impossible to expect the performance of a wide area network. Therefore, the traditional query processing which is based on statistics computed off-line and the static optimization approach [6] is not any more suitable. Furthermore, user's requirements for the query processing may be different from one to others. One may want to get results in a brief delay event if they are not complete, others want to wait for complete and exact results, and others wish to have an approximate answer. In some contexts such as electronic market, digital library, etc., accesses to information are sometimes not free. Users can wish to limit the economic access cost of their query processing. However, it is sometimes difficult to take such decisions before the evaluation of queries because users might have few knowledge of data they query. Therefore, it might be beneficial if users can interact with the query processing. They get first results and refine some criteria of their processing queries or refine their queries.

We claim that there is a need for techniques that allow a distributed query processing taking into consideration the execution environment, i.e. user's requirements, resources constraints, wide area communication, distribution and autonomy of sources, etc., and interacting with this environment while executing queries. Such techniques allow efficiently evaluating queries, i.e. satisfying not only common requirements of query applications but also context-specific, in large-scale mediation systems.

A. Contribution of the Paper

This paper focuses on the query evaluation task in large distributed mediation systems having a high degree of distribution and autonomy of their component sources. It is necessary to go beyond what has been proposed and developed in the context of query processing in distributed database management systems and integration systems, i.e. dynamic optimization, caching, etc. [12], [15], [2], [29].

We propose *Query Brokers* as basic units for evaluating queries. The query evaluation process can be compared as a

Vu Tuyet Trinh, LSR-IMAG Laboratory, BP 72, 38402 St-Martin-d'Hres Cedex, France (Email: Tuyet-Trinh.Vu@imag.fr)

Christine Collet, LSR-IMAG Laboratory, BP 72, 38402 St-Martin-d'Hres Cedex, France (Email: Christine.Collet@imag.fr)

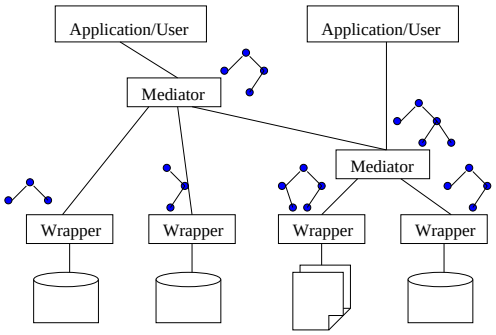

Fig. 1. Hierarchical Mediation Architecture

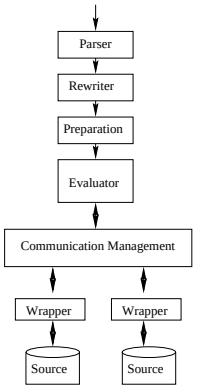

Fig. 2. Query Processing Architecture

set of query brokers. Each of them ensuring the evaluation of a sub-query. The execution context is specified through Query Brokers in order to take into consideration specific requirements while processing queries and to fulfill system available resources [1]. This provides a mean to adapt the query evaluation process to context-specific. Besides, adaptivities of the query evaluation task are enabled by interactions of Query Broker with its execution environment, i.e. users, execution circumstances, during the evaluation phase.

B. Organization

This paper is organized as follows. Section II gives our working hypotheses. Section III presents *Query Broker* and its functional architecture. Section IV shows how the query evaluation based on Query Brokers is flexible. We discuss related works in section V. Finally, section VI concludes the paper.

II. Working Hypotheses

We assume a mediator-based system, i.e. a set of wrapped sources and a mediator with the task of responding to queries formulated on its global schema by using underlying sources. Mediators can be hierarchically organized as shown in Figure 1. Following this approach, a mediator is built on other mediators and/or wrappers. Many mediators can work together to respond to different queries. This approach is suitable to build large-scale systems where mediators can be distributed over network. However, communications between mediators must be taken into consideration while processing queries. As a result, the query processing is distributed through mediators hierarchies. This hypothesis allows us to generalize the mediation architecture, and also the query processing architecture.

As mentioned previously, the static optimization approach is not suitable for processing queries in distributed and scalable mediation systems because of lack of statistics and unpredictabilities of execution environment. Consequently, the query optimization task must be repeated during the execution phase in order to allow multiple modifications of the query evaluation task. For this purpose, we consider a query processing architecture (cf. Figure 2) including the following phases: (i) a parsing phase (*parser*) which syntactically and semantically analyses queries. This phase is similar to the one of the traditional query processing; (ii) a rewriting phase (*rewriter*) which translates global query into local queries on sources schemas. This phase depends on the way mappings between schemas are defined [3]; (iii) a preparing phase (*preparation*) which generates a query evaluation plan (*QEP*). We will present the form of QEP in the next section; (iv) a evaluation phase (*evaluator*) which communicates other components, i.e. mediators or wrappers, for evaluating sub-queries. Please note that the optimization and execution tasks are not separated. They are included into the evaluation phase.

Our work focuses on the evaluation phase. We suppose that there exists an algorithm such as the one in [17], [20] finding relevant sources of a given global query and rewriting the global query into local queries. As a result, we have a query formulated on local schemas terms and our effort aims at evaluating this query on distributed and autonomous sources in a flexible way.

III. Query Broker for Evaluating Queries

To achieve a distributed and flexible query evaluation for large-scale mediation systems, we propose *Query Brokers* as basic units for evaluating queries. Such brokers define (i) the query execution context and (ii) the behaviors of query evaluation towards execution circumstances. Considering the query processing presented in Figure 2, query brokers concern two latest phases, i.e. *preparing* which generates a query brokers graph and *evaluating* which evaluates a query represented by the query brokers graph.

This section presents the role of Query Broker for evaluating queries and its functional architecture.

A. Query Broker

Figure 3 presents *Query Brokers* (QBs) which *wrap one or several query operators*. In other words, a QB corresponds to a sub-query and is the basic unit of the query evaluation. As a result, a QEP is represented as a QBs graph such as the one in

[1] System available resources mean CPU time, memory, etc. which can be used for processing queries.

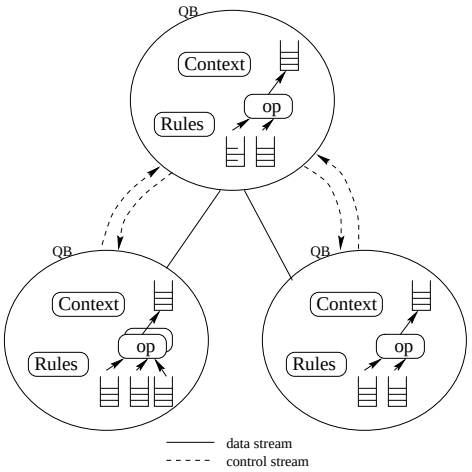

Fig. 3. Interconnected Query Brokers

Figure 3. Our hierarchy of QBs fits the hierarchical mediation architecture presented in Figure 1. Each mediator corresponds to one or several query broker(s) processing sub-query(ies).

A Query Broker is defined by:

- a *context* which determines constraints for executing query and meta-information driving evaluation tasks, i.e. optimization, execution of sub-query wrapped by this broker and communication with other QBs. Examples of constraints are limitation of execution time, acceptation of partial results, limitation of economic access cost, etc.
- *operator(s)* which determines how data is processed by this QB. Operators can be *built-in operators*, i.e. pre-defined operators such as algebraic operators -e.g. `selection`, `projection`, `join`, `union`, etc.-, communication operators -e.g. `send`, `receive`, etc.-, and *user-defined* or *external operators*.
- *buffer(s)* which are used for separating data stream between two QBs. Buffers can have different forms, from simple buffers for synchronizing two QBs operating at different speeds to more or less complicated caches for materializing intermediate results. More details of the buffer management will be discussed in the next sub-section.
- *rules* (`E-C-A`[2]) which define behaviors of QB towards changes of execution environment, e.g. delay of arrival data, inaccessible data, query refinement, etc. Using rules, QBs could change evaluation strategies, e.g. re-schedule/re-optimization sub-queries, change the implementation of certain operators such as join, etc., and behave towards query refinements during the execution phase.

In summary, Query Brokers decouple the execution of several operators. This allows them working in parallel as shown in [8]. Moreover, QBs are basic units for evaluating queries and can be considered as points for caching intermediate results. Thus, they can be used for global query optimization, i.e. optimization of many queries at once. Sub-queries results can be shared among many queries so as to ameliorate the global performance.

[2]When **E**vent If **C**ontition Do **A**ction

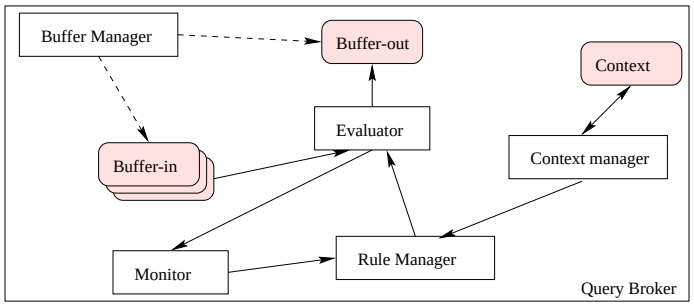

Fig. 4. Functional Architecture of a Query Broker

B. Functional Architecture of a Query Broker

Figure 4 gives an overview of the functional architecture of a QB. The main modules are a *Buffer Manager*, a *Context Manager*, a *Rule Manager*, an *Evaluator*, and a *Monitor*. In the following, we discuss these modules.

a) Buffer Manager: Buffers contain data elements which are inputs or outputs of query operators. Buffers can have different strategies for adding, deleting or replacing their data elements. These strategies depend on the nature of input/output data streams, i.e. "on demand" or continuous data streams, the performance of site locating this broker and connected sites, etc. Buffers can adopt either *pull mode* or *push mode*. Push mode means, the data stream between brokers is activated by children brokers. Children brokers push data to its parent. Pull mode means, parent brokers call data from their children. As a result, our execution model is not limited to only the classical iterator model [9] but apt to different natures of data sources like data sensors [18].

Please note that buffers sizes and buffers strategies can be dynamically modified according to changes of the execution environment. *Buffer Management* is responsible for instantiating QB buffers, managing buffers properties and strategies.

b) Context Manager: As already mentioned, every QB has a context which defines the setting for the evaluation task. Such a QB context depends on available resources, and application requirements. *Context Management* module provides tools for defining and modifying the context. It also checks the changes of context in order to ensure its coherence.

c) Operator Evaluator: This function corresponds to the *Execution Engine* of a classical query processor [6]. It uses a library of query operators including adaptive operators. The adaptive operators are non-blocking operators such as double-pipelined hash join [28], XJoin [25], etc. We also consider some special operators such as `XUnion` - exclusive union - which is designed for handling the case of duplication. `XUnion` produces at output the data coming from only one of its inputs. It is different from the classical *union* operator whose output is collection of data coming from all of its input. The `XUnion`

allows us to model the union of data in the case of duplication. As a result, making choice of data sources can be delayed until execution time according to execution circumstances. This is important in the case of a large distributed integration system because of high degree of duplication of data and unpredictable execution environment.

User-defined functions are considered as part of query operators.

d) Monitor: It is one of the most important module of QB. It ensures the iteration of query processing. It observes the *Evaluator* in order to detect significant changes and adapt query execution to these changes. It also communicates with users and other brokers so as to adapt query execution to changes of the execution environment.

Monitoring can be seen as a learning process that allows updating meta-information collected by system[3] in order to ameliorate query execution in the future and adapting query execution to the environment.

e) Rule Manager: If a QB context defines the setting of QB, i.e. static properties, E-C-A rules define its behaviors, i.e. dynamic properties, dealing with changes of the execution environment.

We define *built-in events* (E) and *actions* (A) that are the minimum set of events and actions supported by the system. Other events and actions can be defined by users[4]. Rules are defined using these events and actions. We will illustrate events, actions and rules in the next section.

Rules can be activated by events coming from the *Buffer Manager*, *Context Manager*, or *Monitor*.

IV. Flexible Query Evaluation

This section shows how Query Brokers support the flexibility of query evaluation. As we already said, flexibility means that the query evaluation process should be easily and dynamically modified to respond to altered execution circumstances or conditions. Therefore, this section comes back to the execution context part and the rule definition part of query brokers.

A. Dynamic Execution Context

Recall that, our goal is not only satisfying common requirements of query applications, but also responding to application context-specific requirements. This requirements are defined through our QB context. The QB context consists of a set of parameters. We determine four categories of parameters related to *user's requirement*, e.g. limitation of execution time (`timeout`), type of partial result (`partial-result`), limitation of economic cost for processing queries (`cost`), interested data (`preference`), etc.; *availability of resources*, e.g. `memory-size`, `CPU-time`, etc.; *meta-information*, e.g. `arrive-data-rate`, `source-access`, etc.; and other query *variables* which will be specified during query execution.

Let us underline that QB context is *query-able*, i.e. users can query QB context as data. In querying QB contexts, users can refine theirs queries, e.g. specify query variables, add new filters, handle process at a QB in order to build partial results, etc., and redefine criteria to process queries, e.g. accept partial results, modify their economic access cost, etc. As a result, the QB context is dynamic. The modification of QB contexts parameters can also be ensured by interactions among QBs in the interconnected QBs graph (Figure 3). These interactions allow the system to "learn" about sources and refresh their meta-information which may not be any more up to date.

B. Rule-based Approach for Dynamic Evaluation Strategies

For achieving a flexible query evaluation framework, we adopted rule-based approach for defining behaviors of QB. E-C-A rules allow to specify Query Brokers behaviors towards execution circumstances. The techniques for re-scheduling and re-optimizing queries [2], [26], [25] can be integrated in QBs as rules.

We distinguish two types of events which are *built-in event* and *user-defined event*. Built-in events are primary events supported by system. We classify *built-in events* into three categories: *feedback event*, *controlled event*, and *internal event*. Feedback events come from QBs producing data, i.e. children QBs in a QBs graph, and aim to propagate changes during the evaluation phase. Controlled events come from QBs which consume data, i.e. QBs parents in a QBs graph, and aim at modifying QBs children context, etc. Internal events do not participate directly in the communication between QBs as feedback and controlled events but they make changes in the context and activate rules that may be origin of other feedback or controlled events. Examples of built-in events are `changed-arrival-data`, `unavailable-source`, `timeout`, etc.

Similarly, there are two types of actions which are *built-in action* and *user-defined action*. We classify built-in actions into two categories: one aims to adapt the context to execution circumstances in order to ameliorate global performance, i.e. modify the parameters of Query Broker, e.g. buffer size, execution time, etc; the other aims to adapt execution strategies, e.g. join methods, parallelization, execution operators order. Examples of built-in actions are `re-parallel`, `re-schedule`, `re-optimize`, `update-context`, etc.

As already mentioned, rules are E-C-A rules of the form: *When* `Event`, *If* `Condition`, *Do* `Action`. Defining a rule means to describe its E-C-A components. Examples of rules are:

```
Rule 1: WHEN    changed_Arrival_Rate(S)
        IF      schedulable()
        DO      re_schedule()
Rule 2: WHEN    changed_Arrival_Rate(S)
        ON      important_change()
        DO      update_meta()
Rule 3: WHEN    timeout()
        DO      return_patrial_result()
```

The first rule is triggered by `changed_Arrival_Rate` event that represents the change of arrival data rate. If the operators execution of this QB is able to re-schedule (`schedulable`), it tries to re-order the execution of query operators. The second

[3]System means the implementation of query evaluation framework based on Query Broker

[4]Users are human being or application using our query evaluation framework

rule is triggered by the same event. If an important change in arrival data rate is detected, the meta-information is updated. The third rule allows the system deciding to build and return partial results when a source is inaccessible (`timeout` event) in order to avoid blocking the query processing. Many rules can be triggered at the arrival of an event, e.g. rule 1 and 2. These rules can be executed in parallel, in consequence or only one (or several) rules will be executed. We use a parametric execution model such as [21] for ordering inter-related rules triggered at the same time.

Recall that, users can define theirs own events and actions. User-defined events/actions can be defined as a composition of built-in events/actions or by users them-selves. Rules are defined on set of built-in and user-defined events/actions. This allows the query evaluation satisfying application context-specific.

V. Related Work

Many works on adaptive query processing have been done such as [8], [23], [16], [13]. Nevertheless, these works only focussed on some specific problems of an adaptive query processing.

The work in [8] addressed the problem of parallelization in query processing distributed over many sites performing at different rate. The `Exchange` operator is added between operators to separate their execution. Consequently, these operators can work in parallel. Using our QB buffers are similar to using the `Exchange` operator. However, providing different strategies for QB buffers (cf. subsection III-B), our execution model is not limited to the classical iterator model in [8]. This allows us to adapt our execution model to different context in a large-scale mediation system.

Our Query Broker framework looks like the one proposed in [23]. However, instead of a centralized "bidder", this evaluation is distributed through an interconnected QBs graph. Besides, [23] focuses on an economic model for optimizing queries while we do not stress on a specific model. Our query evaluation plan is a graph of Query Brokers which define constraints for executing queries. The query execution phase is driven by these constraints. As a result, our solution can satisfy not only common requirements of query applications but also context-specific.

Adaptive techniques such as [2], [1], [24], [14] have been proposed for deciding the best way to execute a query faced on changes of environment during query execution. Several works, such as [2], [26], focus on scheduling query operators faced on delays. The others [14], [25] concentrate to develop auto-adaptive operators, i.e. non-blocking operators. These techniques can be easily integrated in query brokers.

Moreover, different from all of above works, our goal is not only proposing new techniques for adaptive query processing in different contexts but also providing a framework to integrate these techniques. This framework helps to build query applications having many common requirements, and satisfying context-specific requirements. We believe that our QB framework can facilitate the construction of such an application.

VI. Conclusion

This paper presented the definition and role of *Query Brokers* for evaluating queries. Query brokers aim to make a setting (*contexts*) for the query execution to separate the execution of operators, to make them work in parallel, to monitor the execution of queries and to adapt the query execution to execution circumstances. We have also discussed how query brokers ensure the distribution and the flexibility of the query evaluation task in distributed and scalable mediation systems. Using query brokers, the query evaluation task is interactive with its execution environment. During evaluation phase, the system (i) receives information from its environment (*events*), (ii) uses this information to determine its behavior (*rules*), (iii) processes iterates overtime, generates a feedback loop between environment and behaviors (context and rules). These three tasks of our evaluator component fulfill the features of an adaptive query processing defined in [11]. We believe that our approach is suitable for executing queries in the unpredictable environment of large-scale mediation systems.

Acknowledgements

We would like to thank Beatrice Finance for many useful discussions about our work.

References

[1] Ron Avnur and Joseph M. Hellerstein. Eddies: Continuously adaptive query processing. In *SIGMOD Conference 2000*, 2000.
[2] Luc Bouganim, Franoise Fabret, C. Mohan, and Patrick Valduriez. Dynamic query scheduling in data integration systems. In *ICDE*, 2000.
[3] D. Calvanese, D.Lembo, and M.Lenzerini. Survey on methods for query rewriting and query answering using views. Technical report, Technical Report D2I, 2001.
[4] Michael J. Carey, Laura M. Haas, Peter M. Schwarz, et al. Towards Heterogeneous Multimedia Information Systems: The Garlic Approach. In *RIDE-DOM*, 1995.
[5] Ruxandra Domenig and Klaus R. Dittrich. An Overview and Classification of Mediated Query Systems. *SIGMOD Record*, 28(3), 1999.
[6] Elmasri and Navathe. *Fundamentals of Database Systems*, chapter 16, Query Processing and Optimization. 1994.
[7] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, et al. The TSIMMIS Approach to Mediation : Data Models and Languages. *Journal of Intelligent Information System (JIIS)*, 8(2), 1997.
[8] Goetz Graefe. Encapsulation of parallelism in the volcano query processing system. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, May 23-25, 1990*. ACM Press, 1990.
[9] Goetz Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys*, 25(2), 1993.
[10] J. Hammer, H. Garcia-Molina, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. Information translation, mediation, and mosaic-based browsing in the tsimmis system. In *Exhibits Program of the Proceedings of the ACM SIGMOD International Conference on Management of Data*, San Jose, California, June 1995.
[11] Joseph M. Hellerstein, Michael J. Franklin, Sirish Chandrasekaran, Amol Deshpandea, Kris Hildrum, Sam Madden, Vijayshankar Raman, and Mehul A. Shah. Adaptive query processing: Technology in evolution. *IEEE Data Engineering Bulletin*, 23(2), 2000.
[12] Yannis Ioannidis. Query Optimization. *ACM Computing Surveys*, 1996.
[13] Zachary G. Ives, Daniela Florescu, Marc Friedman, et al. An Adaptive Query Execution System for Data Integration. In *SIGMOD Conference*, 1999.
[14] Zachary G. Ives, Alon Y. Levy, Daniel S. Weld, Daniela Florescu, and Marc Friedman. Adaptive Query Processing for Internet Applications. *IEEE Data Engineering Bulletin*, 2000.
[15] Donald Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys*, September 2000.
[16] Le select. http://www-caravel.inria.fr/LeSelect/.

[17] Alon Levy, Anand Rajaraman, and Joann J. Ordille. Query-Answering Algorithms for Information Agents. In *AAAI/IAAI*, volume 1, 1996.

[18] Samuel Madden and Michael J Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In *ICDE*, 2002.

[19] Hubert Naacke, Olga Kapitskaia, Antony Tomasic, Philippe Bonnet, Louiqa Raschid, and Remy Amouroux. The distributed information search component (disco) and the world wide web. In *Proc. of ACM SIGMOD Conf. on Management of Data*, 1997.

[20] Rachel Pottinger and Alon Levy. A Scalable Algorithm for Answering Queries Using View. In *VLDB Conference*, Cairo, Egypt, 2000.

[21] Helena Grazziotin Ribeiro. *Un service de règles actives pour fédérations de bases de données*. PhD thesis, Université Joshph Fourier, 2000.

[22] Mary Tork Roth, Fatma Ozcan, and Laura M. Haas. Cost models do matter: Providing cost information for diverse data sources in a federated system. In *VLDB*, pages 599–610, 1999.

[23] Michael Stonebraker, Paul M. Aoki, Robert Devine, Witold Litwin, and Michael A. Olson:. Mariposa: A new architecture for distributed data. In *Proceedings of 1994 IEEE 10th International Conference on Data Engineering*, Houston, TX, USA, 1994.

[24] Tolga Urhan and Michael Franklin. Xjoin: Getting fast answers from slow and bursty networks. Technical Report CS-TR-3994, 1999.

[25] Tolga Urhan and Michael J. Franklin. Xjoin: A reactively-scheduled pipelined join operator. *IEEE Data Engineering Bulletin*, 2000.

[26] Tolga Urhan, Michael J. Franklin, and Laurent Amsaleg. Cost based query scrambling for initial delays. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*. ACM Press, 1998.

[27] Gio Wiederhold. Mediator in the Achitecture of Future Information systems. *The IEEE Computer Magazine*, 25(3), 1992.

[28] Annita N. Wilschut and Peter M. G. Apers. Dataflow query execution in a parallel main-memory environment. In *Proceedings of the First International Conference on Parallel and Distributed Information Systems (PDIS 1991), Fontainebleu Hilton Resort, Miami Beach, Florida, December 4-6, 1991*. IEEE Computer Society, 1991.

[29] Qiang Zhu and Per-Åke Larson. Solving local cost estimation problem for global query optimization in multidatabase systems. *Distributed and Parallel Databases*, 6(4):373–421, 1998.