Query Trading in Digital Libraries

Fragkiskos Pentaris and Yannis Ioannidis

Department of Informatics and Telecommunications,
University of Athens, Panepistimiopolis, 15771, Athens, Greece
{frank, yannis}@di.uoa.gr

Abstract. In this paper we present a distributed query framework suitable for use in federations of digital libraries (DL). Inspired by e-commerce technology, we recognize CPU-processing and queries (and query answers) as commodities and model the task of query optimization and execution as a task of trading CPU-processing, queries and query-answers. We show that our framework satisfies the needs of modern DL federations by respecting the autonomy of DL nodes and natively supporting their business model. Our query processing conception is independent of the possible distributed architecture and can be easily implemented over a typical GRID architectural infrastructure or a Peer-To-Peer network.

1 Introduction

Digital Libraries' users may need to simultaneously use two or more libraries to find the information they are looking for. This increases the burden of their work as it forces them to pose the same query to different DLs multiple times, each time using a possibly different user interface and a different metadata schema. Digital Libraries are aware of this deficiency and have been trying for a long time to attack this problem by creating federations of DLs. Especially for small DLs, joining such federations is necessary for attracting more users and thus, ensuring their economic survival.

The architectures of these federations usually follow a wrapper-based centralized mediation approach. Nevertheless, the growth of DL collections and the increase in the number of DLs joining federations have rendered these architectures obsolete. Almost every major DL is evaluating new architectures to replace its old systems. For instance, a kind of P2P architecture will be evaluated within the framework of the BRICKS [2] European Integrated Project (peer nodes are called Bricks nodes in this project), whereas the GRID architecture will be evaluated within the DILIGENT [8].

Obviously, existing DL federations will benefit a lot by the above architectures as the improved search and browse response-time will enable them to form even larger federations. On the other hand, even today's hardware and software architectures (e.g., ultra fast SANs) do provide the means for building fast federations, yet DLs are still reluctant in unconditionally joining them. It seems that apart from the scalability problem, there are additional ones inhibiting the creation of large federations. DLs prefer to keep their systems completely autonomous. They want their nodes to be treated as black boxes, meaning that remote nodes should make no assumption on the contents, status and capabilities of their systems. Exporting this information to distant nodes hurts the autonomy of DLs, which in turn reduces their flexibility and increases the burden of their work. An additional problem is that DLs are usually competitive institutions, therefore, the proposed distributed architectures should natively respect and support their business requirements.

The main contribution of this position paper is the presentation of a query processing schema, which may be setup over a P2P or GRID-based network architecture. Our proposal respects the autonomy of existing DLs and natively addresses their business model. The rest of the paper is organized as follows: In section 2, we discuss the business requirements of DLs. In section 3, we present our query processing architecture. In section 4, we discuss any relevant work before concluding.

2 Digital Libraries Federations Requirements

In the introduction we argued that DLs are reluctant in forming federations because they are not sure that these systems comply with their business model and respect their autonomy. In the following paragraphs, we briefly examine these requirements focusing on the problems they create to the two most prominent future DL architectures, i.e, P2P networks and the GRID.

Business Model - Content Sharing. Economic prosperity of Digital Libraries is usually bound to their ability to sell information (content, annotation and metadata) and data processing services to their users. These are in fact the only assets DLs hold, making them reluctant to give away any data to third-party entities, especially if this is done over the Internet. For instance, in BRICKS many institutions will not export or mirror their data to the common BRICKS community network but instead will allow access to their data and legacy systems through the use of conventional wrappers. Employing a strict security and trust policy in every network node and using state-of-the-art content-watermarking techniques reduces the reluctance of DLs in sharing their data. Nevertheless, experience shows that no security system is perfect and DLs are aware of this fact.

The reluctance of DL to share their content creates a lot of problems in architectures following the GRID paradigm, since the latter model the process of evaluating queries as a task of moving the data (content) to one or more processing GRID-nodes, and then starting the actual execution of these queries. Obviously, a completely new query processing architecture must be used that will minimize the physical movement of (unprocessed) data.

P2P-based systems are also affected by the content sharing restriction problem. Building a metadata P2P indexing service, using a Distributed Hash Table (DHT), will distribute the metadata of a DL to multiple, potentially less trusted nodes, including other competitive DLs. This is not something that a DL's manager will easily approve. A solution would be to use a double hashing technique, i.e., build a DHT of the hash value of the metadata instead of the metadata themselves. In this way, nodes will know the hash value of the metadata but not the exact metadata. If the users make only keywords-based queries, this approach will be satisfactory. But if advanced query capabilities are required, a traditional query processor that uses DHT indices and requires the physical movement of data, just like the GRID architecture, will have to be used. Thus, even in the case of P2P-based systems, a new query processing framework is required that will also minimize the physical movement of unprocessed information.

Business Model - Integration of Query Processing and Accounting. Consider a small federation of two DLs where the first DL holds digital pictures while the second one has information concerning poetry. Assume that a user of this federated system is looking for pictures that were created by painters that have also written certain types of poems. This query combines pieces of information from both DLs, yet only retrieves/browses content from the first one. Since DL sell (any possible piece of) information, it is a matter of the billing policy of each DL, whether the user should be charged only for the retrieved content, or also for the information of the second DL that was used during query processing. If DLs choose to charge for any information they provide, which we expect in the future to be a typical business scenario, the query optimizer and the accounting system should be closely integrated.

Competitive Environment. The most important business requirement of future DLs federations is that these should be compatible with the competitive nature of the DLs market, i.e., information is asset and data should be treated as commodities for trading in a competitive environment. Competition creates problems in DL federations, as it results in potentially inconsistent behavior of the nodes at different times. The query processing architecture should be capable of handling cases where remote nodes expose imprecise information. Such behavior is typical (and allowed) in competitive environments.

Autonomy. A requirement of modern DL federations is that distributed architectures should respect the autonomy of DLs and treat them as black boxes. Middle-wares and wrapper-based architectures help in preserving the autonomy of remote nodes. However, during query processing and optimization, existing proposals require, *a priori*, certain pieces of information (e.g., data statistics, remote nodes status (workload), nodes capabilities (e.g., which variables must be bound), operators (e.g., union, join) cost functions and parameters) that clearly violate their autonomy. A proper query processing architecture that respects DL's autonomy and work only with information that nodes expose during query processing.

3 The Query Trading Architecture

3.1 Execution Environment

We have recently proposed a new query processing architecture [18] that meets the scalability and autonomy requirements of future DLs and perfectly matches their business requirements. Figure 1 presents a typical example of the proposed architecture. It shows a network of five autonomous DL nodes ($N_1 - N_5$). Each of them, stores its own information (data and metadata) and may additionally store copies of other nodes' data and metadata, if such a thing is allowed by the policy of these distant nodes. For instance, in Fig. 1, both nodes N_2 and N_3 have information on "Greek Documents" for the period 500BC-400BC. This redundancy may be necessary for load-balancing or robustness reasons, or it may be simply the natural result of competition between nodes N_2 and N_3.

The only requirement of our architecture is that there must be a kind of directory service holding information on which data each node locally holds. For small networks,

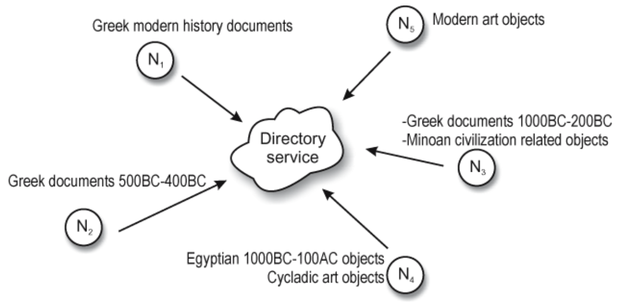
Greek modern history documents
N1
N5
Modern art objects
Directory service
-Greek documents 1000BC-200BC
-Minoan civilization related objects
N3
Greek documents 500BC-400BC
N2
Egyptian 1000BC-100AC objects
Cycladic art objects
N4

Fig. 1. Architecture overview

this can be implemented using a centralized mechanism (e.g. an LDAP server). For larger networks, the directory service can follow the P2P and DHT paradigm. Each node must register in the directory service a (high-level) description of the data it holds. In Fig. 1 we have drawn the contents of the directory service next to each node. We should note that the directory service is used only in the initial phase of queries' evaluation to locate relevant (to the queries) DL nodes. The selection of the actual nodes that will evaluate these queries is handled by the economics-based mechanism presented in the next subsection. This is different to a traditional P2P search engine. The latter contains information on all the objects (and their properties/attributes/terms) of the distributed system, whereas our directory service contains more high-level information such as which nodes hold information concerning (e.g.) middle-age pictures.

3.2 Query Evaluation Mechanism

During query evaluation, we treat DL nodes as black boxes, assuming nothing on their workload or capabilities. The only information required is the one available thought the directory service. Execution of distributed queries is handled by splitting them into pieces (sub-queries), forwarding them to nodes capable of answering them, and then combining the results of these queries to build the answer of the distributed query.

To make our architecture more concrete, continuing the previous example (Fig. 1) we assume that a user at node N_1 asks for the URLs of every ancient Greek document written in Athens between 500BC and 498BC. Since DL nodes are black boxes, N_1 can do nothing better than asking the rest DL nodes for any piece of information that might be of some help in evaluating the query. As Fig. 2 shows, network congestion is avoided by having N_1 sent its "request for help" only to the directory service, which in turn, only forwards this request to the relevant (to the query) nodes. Any node answering to this request sends its reply directly to the initial node (N_1), which further reduces network bandwidth consumption.

In the example of Fig. 2, we assume that N_2 offers to return to N_1 a URL list of all relevant documents written between 500BC and 499BC in 25 seconds and the rest documents (498BC) in 20 seconds. Similarly, node N_3 offers the same information in 50 and 15 seconds respectively. Obviously, node N_1 query optimizer will choose node N_2 for all URLs concerning documents written in 500-498BC and N_3 for the rest ones.

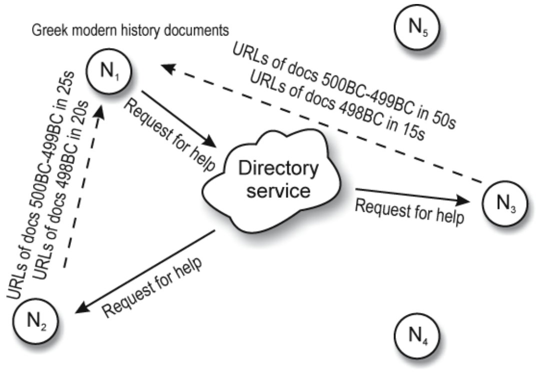
Greek modern history documents
N1
N2
N3
N4
N5
Directory service
Request for help
Request for help
Request for help
URLs of docs 500BC-499BC in 50s
URLs of docs 498BC in 15s
URLs of docs 500BC-499BC in 25s
URLs of docs 498BC in 20s

Fig. 2. Example of query processing

That is, the query processor of N_1 effectively purchases the answer of the original query from nodes N_2 and N_3 for 20 and 15 seconds respectively.

The above example shows the main idea behind the query processing architecture that we propose. It is inspired by e-commerce technology, recognizes queries (and query answers) as commodities and approaches DL federations as information markets where the commodities sold are data. Query parts (and their answers) are traded between DL nodes until deals are struck with some nodes for all of them. Distributed query execution is thus modeled as a trading of parts of query-answers. Buyer nodes (e.g., N_1) are those requiring certain pieces of information in order to answer a user query. Seller nodes (e.g., N_2 and N_3) are those offering buyers this missing information.

Although the idea is simple, it is difficult to construct an algorithm that can optimize the trading of queries and queries answers. For instance, assume that in the previous example, node N_3 offered the URLs of the documents written in 500BC,499BC and 498BC separately for 20s, 30s, and 15s respectively. In this case, node N_1 has many different ways of combining the offers of N_2 an N_3. In fact, it might worth for N_1 to negotiate with node N_2 the case of N_2 also returning the URLs of the documents written in 500BC and 499BC separately, before node N_1 decides on which offers can be combined in the best way.

3.3 General Trading Negotiation Parameters

There are a lot of parameters that affect the performance of a trading framework such as the one described in the previous sub-section. For details on these parameters see [1,3,4,11,13,15,17,18,21,22,25]. We briefly describe the most important ones:

Negotiation protocol. Trading negotiation procedures follow rules defined in a negotiation protocol [25]. In each step of the procedure, the protocol designates a number of possible actions (e.g., make a better offer, accept offer, reject offer, etc.) that a node may take. In the previous example, we assumed that the protocol used was *bidding* [23]. This protocol is simple but obviously cannot work when the number of nodes is large. In larger networks, plain bidding would lead to network flooding problems. In this case, a better alternative is to use an agent-based or P2P-based [15] *auction*, which reduces the number of exchanged network messages. If the items/properties negotiated are minor and the nodes participating in the negotiation are few, then the oldest known protocol, *bargaining* can be also used.

Strategy. In each step of the negotiation procedure and depending on the negotiation protocol followed, nodes have multiple possible actions to choose from. It is the *strategy* followed by each node that designates which action is the best one. The strategy can be either cooperative or competitive (non-cooperative). In the first case, nodes try to maximize the join utility of all nodes that participate in the negotiation. In the second case, nodes maximize their personal utility. Our architecture supports both types of strategies. In cooperative ones, nodes expose information that is accurate and complete. In competitive setups, nodes expose information that is usually imprecise. For instance, a node may lie about the time required for the retrieval of a piece of information.

User preferences and items valuation. In section 3.2 we gave an example where the value of the commodities (i.e., the pieces of information) offered by remote nodes was expressed in term of the time required to fetch this information. More generally, offers of remote nodes will have many different properties, including (e.g.) the time required to retrieve the information, the precision and age of the data, and its cost in monetary units. That is, the valuation of an offer is multi-dimensional (a vector of values). The user must supply a preference relation over the domain of these vectors that orders the set of possible offers. This relation is known to buyer nodes and is used during the negotiation phase (e.g., during bidding) to select the offers that best fit the needs of the user.

Market Equilibrium. In competitive environments, nodes provide imprecise information. For instance, if the preference relation is the total cost (in monetary units) of the answer, then nodes will increase the value of all pieces of information that have more demand than supply. This will cause a decrease in the demand of this information and after some time, values will stop fluctuating and the market will be in equilibrium, i.e., demand will equal supply for all traded queries. This requires all other parameters affecting the value of items to be static [6]. The designer of a system can model the market in such a way that equilibrium values force the system to have a specific behavior (e.g, altruistic nodes are not overloaded). A nice property of our architecture is that according to the first welfare theorem [20] of economics, equilibriums will always be Pareto optimal, i.e., no node can increase its utility without decreasing the utility of at least on other node.

Message congestion mechanisms. Distributed implementation of the previous negotiation protocols have run into message congestion problems [23] caused by offers flooding. This can be avoided using several different approaches such as agent-based architectures, focused addressing, audience restriction, use-based communication charges and, mutual monitoring [17,23].

3.4 The Proposed Architecture

As it was mentioned earlier, we model query processing as a query trading procedure. Although there is a lot of existing work in e-commerce and e-trading (see previous subsection), there in an important difference between trading queries (and their answers) and the rest commodities. In traditional e-commerce solutions, the buyer receives offers for the complete items that he/she has asked for. However, in our case, it is possible that no DL node has every piece of information required to answer a user supplied query. Sellers will have to make offers for parts of the query (sub-queries) depending on the information that each DL holds locally. Buyers will have to somehow merge these offers to produce the answer of the initial queries. Since all nodes are black boxes, most sellers will make overlapping offers and buyers will have to make multiple rounds of communication with the seller nodes to ensure that the accepted offers are not overlapping. The problem of query optimization also complicates the task of the buyer since better offers not always improve the global distributed query execution plan. In the next paragraphs, we present how query optimization works in our framework. Further details on the proposed framework and its performance characteristics are given in [18].

The distributed execution plans produced by our framework consist of the query-answers offered by remote DL seller nodes together with the processing operations required to construct the results of the optimized queries from these offers. The query optimization algorithm [18] finds the combination of offers and local processing operations that minimizes the valuation (cost) of the final answer. For this reason, it runs iteratively, progressively selecting the best execution plan. In each iteration, the buyer node asks (Request for Bids -RFBs) for some queries and the sellers reply with offers that contain the estimations of the properties of these queries (query-answers). Since sellers may not have all the data referenced in a query, they are allowed to give offers for only the part of the data they actually have. At the end of each iteration, the buyer uses the received offers to find the best possible execution plan, and then, the algorithm starts again with a possibly new set of queries that might be used to construct an even better execution plan. The buyer may contact different selling nodes in each iteration, as the additional queries may be better offered by other nodes. This is in contrast to the traditional trading framework, where the participants in a negotiation remain constant.

In order to demonstrate our algorithm, we will use Fig. 3 that shows a typical message workflow among the buyer and seller nodes when the number of nodes is small (i.e., the bidding protocol is sufficient and we don't need to use auctions) and nodes follow a cooperative strategy. In this figure, a node receives a query Q that cannot be answered with the data that this node locally holds. For this reason it acts as a buyer node and broadcasts a RFB concerning query Q to some candidate seller nodes. The sellers in their turn, examine the query and if they locally have any relevant information concerning parts of it, they inform the buyer of the properties of these parts (offers). In our example, only two nodes return some offers back to the buyer.

The buyer node waits for a timer to expire (bidding duration) and then considers all offers it has received to construct an initial optimal distributed query execution plan. It then examines this plan to find any other possible part of the query that could help the buyer further improve the distributed plan. In Fig. 3 we assume that the buyer (e.g.) found some parts of the initial query that are offered by both sellers. For this reason it starts a second iteration of the bidding procedure, this time requesting for bids on these overlapping parts. Figure 3 shows that only one seller makes an offer in the second bidding procedure. After the bidding procedure of the second negotiation is over, the

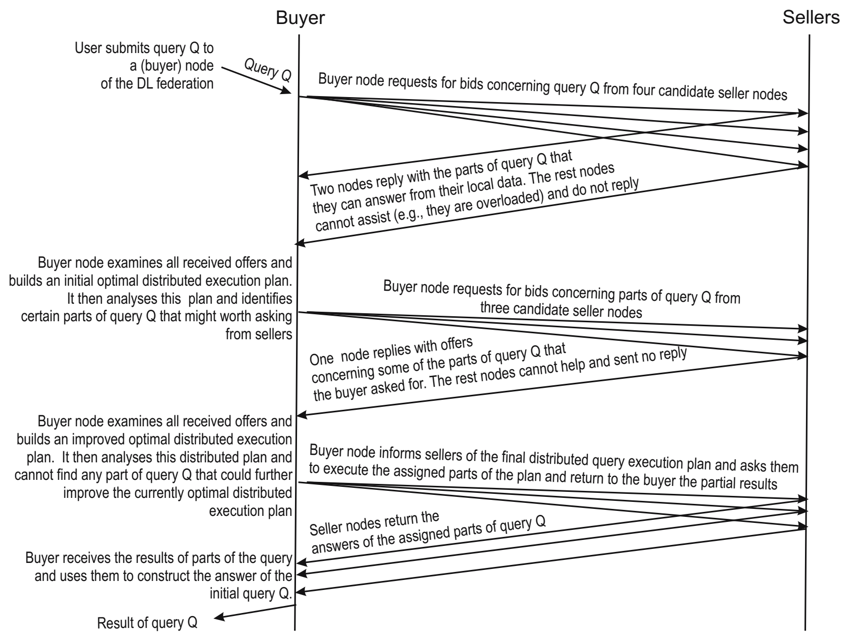
Buyer
Sellers
User submits query Q to a (buyer) node of the DL federation
Query Q
Buyer node requests for bids concerning query Q from four candidate seller nodes
Two nodes reply with the parts of query Q that they can answer from their local data. The rest nodes cannot assist (e.g., they are overloaded) and do not reply
Buyer node examines all received offers and builds an initial optimal distributed execution plan. It then analyses this plan and identifies certain parts of query Q that might worth asking from sellers
Buyer node requests for bids concerning parts of query Q from three candidate seller nodes
One node replies with offers concerning some of the parts of query Q that the buyer asked for. The rest nodes cannot help and sent no reply
Buyer node examines all received offers and builds an improved optimal distributed execution plan. It then analyses this distributed plan and cannot find any part of query Q that could further improve the currently optimal distributed execution plan
Buyer node informs sellers of the final distributed query execution plan and asks them to execute the assigned parts of the plan and return to the buyer the partial results
Seller nodes return the answers of the assigned parts of query Q
Buyer receives the results of parts of the query and uses them to construct the answer of the initial query Q.
Result of query Q

Fig. 3. Workflow of network messages between the buyer and seller nodes

buyer uses the new offer(s) to further improve the distributed plan and then re-examines it to find any other possible part of the query that can be improved. In our example, we assume that the buyer cannot find any such sub-query. Therefore, it asks from the selected remote nodes to evaluate the parts of the distributed plan that have been assigned to them and then return the results of these parts back to the buyer node. The latter uses these results to construct the answer of the initial query Q.

3.5 Processing-Tasks Trading

It is common in DLs to pose queries requiring substantial CPU processing to answer. For instance, a user may ask for all Cycladic-art picture objects that look *similar* to a specific one, where the similarity of two pictures is defined according to a user-supplied function. Answering this query will require substantial processing to evaluate the user-supplied function multiple times. For this reason in [19] we proposed a further enhancement to the previous query trading algorithm (QT) that allows it to also trade *processing-tasks*. There are three different ways to accomplish this:

Single processing-task trading. After the query trading has been completed and a distributed query execution plan has been produced, we can analyze this plan to identify processing-hungry operations that might worth assigning to distant nodes (e.g., a user-supplied picture similarity function, or some CPU-bounded join operators). We could then run a *single* round of processing-tasks trading to assign them to remote nodes.

Table 1. Comparison of different ways of query and process-task trading

Algorithm	Advantages	Disadvantages	Suitability
Plain query trading	Fastest optimization mechanism	Produces the worst query execution plans	Small queries without user-defined functions
Single processing-task trading	Fast optimization mechanism	Limited capabilities for assigning processing tasks	Large queries with non-complex user-defined functions
Iterative query and processing-task trading	Assigns many processing tasks to remote nodes	Slow optimization mechanism	Large queries with complex user-defined functions
Simultaneous query and processing-task trading	Full distribution of processing tasks	Very slow optimization mechanism	Very large queries with heavy processing requirements

Apart from the fact that we trade processing-tasks instead of queries, processing-tasks trading is similar to the plain query trading described in the previous section.
Single processing-tasks trading is especially useful when optimizing DL queries containing processing operators that should be applied after the matching rows/objects have been retrieved. An example of such a query would be the one asking for *thumbnail* pictures of all Cycladic-art objects found in 2004. Note that if this query was expressed in SQL, its *select* part would contain a user-defined function that converts the retrieved full-resolution pictures to thumbnails.

Iterative query and processing-task trading. The second approach is to run a query trading followed by a processing-task trading iteratively, until the distributed query execution plan cannot be further improved. This approach is better than the first one, since it partially integrates query and processing trading. However, it increases the time required for query optimization and thus should be preferred in queries involving processing of user-defined function before the matching DL objects have been identified. If these queries were expressed in SQL, they would have contained user-defined functions in the *where* part of the query.

Simultaneous query and processing-task trading. The last approach is to fully integrate query and processing trading, i.e., simultaneously request bids for both queries and processing. This approach yields the best query execution plans but requires excessive time for the query optimization. Therefore, it should be the preferred option for very large queries requiring substantial amounts of CPU-processing.

Table 1 summarizes the advantages and disadvantages of the three different ways of processing-task trading. This summary is the result of an excessive set of experiments presented in [19].

3.6 Parameters Affecting Query and Processing-Task Trading

In section 3.3 we discussed the parameters affecting all trading frameworks, including the ones presented in this paper. In this section, we focus our attention on the query and processing-task trading specific parameters.

Items valuation. In section 3.3 we argued the the valuation of an offer is multi-dimensional. However, this does not necessary mean that offers, made for a specific query, will differ in many attributes. It can be proved that under certain conditions in competitive (non-cooperative) environments, *if all offers for a query differ in only one property (e.g., query execution time), then in market equilibrium all nodes will make the same offer for that query*. For instance, if the query trading framework is used as a classical query optimization mechanism, then all offers for each query will differ only in the query execution time. Then, the previous proposition states that in competitive market equilibrium, all nodes for the same query will offer the same execution time. The proof of this proposition is based on the fact that (a) only a single equilibrium price exists and (b) no node will have an incentive to deviate from that market price.

Negotiation protocol. In the previous paragraph we argued that if offers differ in only one property, then in the end (at market equilibrium), all offers for the same query will be equivalent. In this case, the best negotiation protocol to use, is plain bidding or auctioning, since all offer's properties are constant and thus non-negotiable. This does not hold if offers may differ in more than one property. For instance, if the query trading framework is used for the purpose of charging users for the search and browse facilities used, then nodes' offers for the same query will differ in (e.g.) both the price and in the quality of the data offered. In this case, the preferences of the user can be better satisfied using a multi-lateral bargaining protocol, that will allow users to efficiently and simultaneously negotiate all possible price-quality combinations.

Strategy. In cooperative environments where offers for the same query differ in only one property, there is no reason to implement any strategy. If the environment is competitive, then results from the theory of games [21] should be used. Finally, the most difficult case is when offers for the same query differ in multiple properties. Then, the resulting trading framework solves the problem of *multi-objective* query optimization. Examples of strategies that work in this scenario are given in [14,7].

Market Equilibrium. In section 3.3 we argued that in market equilibrium, the allocation of resources is Pareto optimal. It can be proved [19] that usually, the requirements of the second theorem of microeconomics [7] hold for the query and processing-tasks trading framework. This theorem is the opposite of the first theorem of microeconomics mentioned in section 3.3 and in our case, states that *any load-balancing algorithm achieving Pareto optimal resource distribution in distributed DLs can be implemented using the query trading algorithm*. This last proposition shows the power of our query and processing-tasks trading algorithm.

Subcontracting. The example of section 3.2 was a rather simple case, since sellers nodes did not considered the case of constructing offers using data retrieved from third party nodes, i.e., subcontracting parts of offers. In experiments presented in [19], it was shown that subcontracting increases network messages exchanges and thus, does not always increase the performance of the distributed system. However, in many cases, this technique is unavoidable as it is the only one allowing buyers to acquire data residing in nodes that are only indirectly (though a third node) accessible to them.

Contracting. Contracts are used in microeconomics to describe the obligations of sellers and buyers and usually define a penalty that a buyer/seller will pay to the seller/buyer if it unilaterally breaks the contract. In the trading framework, contracts can be used to model the notion of adaptive query optimization. As mentioned in section 3.4, the query trading is an iterative algorithm that initially finds and then progressively optimizes the distributed execution plan of queries. We may use the notion of contracting to allow buyers to early start the evaluation of queries (i.e., make an early contract), before the final iteration of the trading algorithm has completed. Early contracting reduces the algorithm execution time, yet, it risks the contracted query execution plan to be much more worsen than the optimal one. In this case, buyers will have to stop the evaluation of the query, wasting a lot of resources (the penalty of breaking the contract), and then restart query evaluation using the optimal execution plan (found in the last iteration of the algorithm). Note that this is one of the ideas behind the notion of adaptive query optimization. Thus, with the proposed contracting modelling, we can use the existing microeconomic theory (e.g., [14]) to predict the performance of this type of adaptive query optimization.

Quality of Service. Previously, we discussed the possibility of a buyer breaking a contact. More generally, the opposite can also happen, i.e, a seller may unilaterally break a contract. For instance, if a network failure occurs, then some sellers may not be able to fulfill their contracts with distant buyers. This possibility is handled in microeconomics using the notion of *insurance*, which can also be used in our trading framework. The role of insurance companies will be played by certain network nodes and links that will be ready to assist in query evaluation if a seller runs into some predefined difficulties (e.g., out of processing resources). Existing microeconomic theory and the theory of *choice under uncertainty* can be used to calculate the exact amounts of resources that must be reserved by the insurance nodes, so that the whole DL network exhibits a certain levels of QoS.

4 Related Work

There is a lot of work in distributed query execution and optimization over P2P systems. However, query-processing techniques employed by these systems cannot be used (directly) in DL systems, as P2P systems are typically/often limited to keyword-based searches, and thus cannot support advance predicate- or ontology-based queries. As far as grid architectures are concerned, DL node autonomy and diversity result in lack of knowledge about any particular node with respect to the information it can produce and its characteristics, e.g., query capabilities, cost of production, or quality of produced results. If inter-node competition exists, it additionally results in potentially inconsistent node behavior at different times. All these problems make traditional query optimization techniques [9,10,16] inappropriate [5,12,24] for autonomous systems such as the ones encountered in Digital Libraries. Our proposed trading framework natively handles these problems without any difficulty.

As far as the authors are aware of, the only architecture that closely resembles ours is Mariposa [24]. Nevertheless, Mariposa's optimization algorithm produces plans that exhibit unnecessarily high communication costs [12] and are arbitrarily far from the desired optimum [16,18]. Furthermore, Mariposa violates the autonomy of remote nodes as it require all nodes to follow a common cost model and asks remote nodes to expose information on their internal state (e.g., their current workload).

5 Conclusion

We propose a query processing paradigm, that respects the autonomy of DL nodes and natively supports their business model (information trading). Our framework natively supports distributed query optimization and allows for Pareto Optimal allocation of DL resources. It can be easily implemented over a typical GRID architectural infrastructure, where the GRID nodes will act as sellers and/or buyers of information and processing. For scalability reasons, a decentralized (P2P) agent-based auction mechanism and/or a P2P DHT for the directory service implementation can be used.

References

1. M. Bichler, M. Kaukal, and A. Segev. Multi-attribute auctions for electronic procurement. In *Proc. of the 1st IBM IAC Workshop on Internet Based Negotiation Technologies, Yorktown Heights, NY, March 18-19*, 1999.
2. BRICKS integrated project. `http://www.brickscommunity.org/`, 2004.
3. John Collins, Maksim Tsvetovat, Rashmi Sundareswara, Joshua van Tonder, Maria L. Gini, and Bamshad Mobasher. Evaluating risk: Flexibility and feasibility in multi-agent contracting. In *Proc. of the 3rd Annual Conf. on Autonomous Agents , Seattle, WA, USA.*, May 1999.
4. V. Conitzer and T. Sandholm. Complexity results about nash equilibria. *Technical report CMU-CS-02-135,* `http://www-2.cs.cmu.edu/~sandholm/Nash_complexity.pdf`, *2002.*
5. Amol Deshpande and Joseph M. Hellerstein. Decoupled query optimization for federated database systems. In *Proc. of 18th. ICDE, San Jose, CA*, pages 716–727, 2002.
6. Donald Ferguson, Christos Nicolaou, and Yechiam Yemini. An economy for managing replicated data in autonomous decentralized systems. In *Proc. of Int. Symposium on Autonomous and Decentralized Systems*, 1993.
7. Hugh Gravelle and Ray Rees. *Microeconomics (3rd edition).* Pearson Education, England, 2004.
8. Diligent integrated project. `http://http://diligentproject.org/`.
9. Yannis E. Ioannidis and Younkyung Cha Kang. Randomized algorithms for optimizing large join queries. In Hector Garcia-Molina and H. V. Jagadish, editors, *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, May 23-25, 1990*, pages 312–321. ACM Press, 1990.
10. Yannis E. Ioannidis, Raymond T. Ng, Kyuseok Shim, and Timos K. Sellis. Parametric query optimization. *VLDB Journal*, 6(2):132–151, 1997.
11. John H. Kagel. *Auctions: A Survey of Experimental Research.* The Handbook of Experimental Economics, edited by John E. Kagel and Alvin E. Roth, Princeton: Princeton University Press, 1995.

12. Donald Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys*, September 2000.
13. Sarit Kraus. *Strategic Negotiation in Multiagent Environments (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2001.
14. Andreu Mas-Colell, Michael D. Whinston, and Jerry R. Green. *Microeconomic Theory*. Oxford University Press, 1995.
15. E. Ogston and Stamatis Vassiliadis. A Peer-to-Peer Agent Auction. In *Proc. of AAMAS02, Bologna, Italy*, July 15–19 2002.
16. Christos H. Papadimitriou and Michalis Yannakakis. Multiobjective query optimization. In *Proc. of the 20th ACM SIGACT-SIGMOD-SIGART Symposium on PODS, May 21-23, 2001, Santa Barbara, CA, USA*. ACM, ACM, 2001.
17. H. Van Dyke Parunak. *Manufacturing experience with the contract net.* Distributed Artificial Intelligence, Michael N. Huhns (editor), Research Notes in Artificial Intelligence, chapter 10, pages 285-310. Pitman, 1987.
18. Fragkiskos Pentaris and Yannis Ioannidis. Distributed query optimization by query trading. In *Proc. of Int. Conf. on Extending Database Technology (EDBT), Herakleio, Greece*, 2004.
19. Fragkiskos Pentaris and Yannis Ioannidis. Query optimization in autonomous distributed database systems. submitted, 2004.
20. Mark Pingle and Leigh Tesfatsion. Overlapping generations, intermediation, and the first welfare theorem. *Journal of Economic Behavior and Organization.*, 3(5):325–345, 1991.
21. J. S. Rosenchein and G. Zlotkin. *Rules of Encounter : designing conventions for automated negotiation among computers*. The MIT Press series in artificial intelligence, 1994.
22. Tuomas Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135:1–54, 2002.
23. Reid G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, 29(12):1104–1113, December 1980.
24. Michael Stonebraker, Paul M. Aoki, Witold Litwin, Avi Pfeller, Adm Sah, Jeff Sidell, Carl Staelin, and Andrew Yu. Mariposa: A wide-area distributed database system. *VLDB Journal*, 5(1):48–63, 1996.
25. Stanley Y.W. Su, Chunbo Huang, Joachim Hammer, Yihua Huang, Haifei Li, Liu Wang, Youzhong Liu, Charnyote Pluempitiwiriyawej, Minsoo Lee, and Herman Lam. An internet-based negotiation server for e-commerce. *VLDB Journal*, 10:72–90, 2001.