Query Processing in Distributed Environments of Autonomous Data Management Systems*

Fragkiskos Pentaris**

University of Athens, Deptartment of Informatics and Telecommunications
frank@di.uoa.gr

Abstract. In this thesis summary, inspired by e-commerce technology, we recognize queries as commodities and model query optimization and allocation as a trading negotiation process.

1 Introduction

Current requirements on scalability and availability of information foster the formation of large networks of databases with large amounts of data distributed among hundreds or even thousands of autonomous and possibly heterogeneous Database Management Systems (DBMSs). In such environments, finding the answer to a query requires splitting it into parts (sub-queries), retrieving the answers of these parts from several remote "black-box" nodes, and merging the results together to calculate the answer of the initial query. This poses significant challenges to query processing and optimization. Autonomy is the main source of the problem, as it results in lack of knowledge about any particular node with respect to the information it can produce and its characteristics, e.g., cost of production or quality of produced results. Earlier work [2, 8] has shown that traditional query optimization techniques do not perform well in federations of autonomous DBMSs. In [6, 7] we further examined these techniques and most of them were found to conflict with DBMS autonomy.

An additional problem related with query processing in disparate systems is that two of their most important components, the distributed query optimizer and the query (resources) allocation mechanism, often conflict with each other. The optimizer tries hard to produce query execution plans that minimize the query response time by selecting the fastest data access and join methods, by maximizing the number of nodes (degree of parallelism) involved in the processing of each query separately, and by making sure that the nodes selected are the fastest available ones. Unfortunately, the first two of these three strategies frequently achieve a Pyrrhic victory; they deplete the resources of the whole distributed system by increasing the average number of resources used per query. At the same time, assigning processing to the fastest of the available nodes may case substantial load imbalance among system nodes. In this case, not only will

* The majority of this work appears in [6, 7] and in [5].
** Desertation advisor: Yannis Ioannidis, Professor

the performance of the whole distributed system suffer, but also the load distribution mechanism will intervene and conflict with the futile decisions of the query optimizer.

In this thesis, we revisit the problem of query processing in autonomous disparate systems, proposing four distributed query optimization and three query allocation mechanisms. Our mechanisms are microeconomics-inspired and especially designed so as to run autonomously and to not conflict with each other. More specifically, we propose a query trading mechanism where instead of trading goods, nodes trade answers of (parts of) queries and operator-execution jobs. The market acts as an integrated distributed query optimization and allocation mechanism. Not only does it find one of the best possible distributed query execution plans for a single query, but also improves query throughput when multiple, concurrent queries are evaluated. Our solution is especially efficient in *autonomous* environments and, as far as the author is aware of, this is the first time that query optimization and allocation is treated in an *integrated way*. It is also one of the few cases that distributed query processing is considered *end-to-end*, starting from the trivial case of optimizing a single query to the real-life case of *concurrently optimizing, allocating and running multiple distributed ones*.

The remainder of this thesis summary goes as follows: In section 2, we formally present the problem solved. In section 3, we describe our basic query optimization and processing framework. In section 4 we present a query allocation mechanism. More details, additional algorithms and expremental analysis can be found in thesis text.

2 Problem Definition

Throughout this thesis, we consider distributed systems consisting of autonomous and possibly diverse database management systems (DBMSs) that are interconnected using a fast, local or wide area, network. Nodes are heterogeneous in terms of their software and hardware capabilities. They may use different DBMS configurations or products possibly supporting different access plans. The memory, I/O and CPU resources offered to distant nodes can vary, dynamically being modified by node administrators. For simplicity, we only consider relational DBMSs, though, many of our findings are data model independent. We allow nodes to act in a cooperative or competitive manner towards each other.

Tables may be mirrored to multiple nodes or be horizontally partitioned, yet, we require all local node schemas to be a subset of a "global" one. Mirrors of non-partitioned tables have the same name and are assumed to hold the same data. We allow (and encourage) partitions of a single table to be displaced. Similarly to the case of non-partitioned tables, we assume that mirrors of the same partition always hold the same data. Each node may have different indexes or materialized views; the solution proposed will automatically use them.

The workload of the distributed system consists of select-join-project queries that reference only global schema tables. A query can be submitted, if allowed, to any node of the network, and any node, if it wants to, can offer to assist in the

evaluation of some parts of this query. The answer is expected to be delivered to the initial node where the query was submitted. We allow any number of queries to be concurrently evaluated. Node autonomy and diversity forces us to treat distant DBMSs as black boxes that can only evaluate queries expressed in SQL. Distributed queries are evaluated by splitting them into pieces (subqueries) expressed in SQL, assigning these pieces to distant nodes and then post-processing and merging the results to produce the answers of the initial distributed queries.

Given the environment just described, our objective is to *construct a distributed algorithm that minimizes the average response time of all concurrently executed queries, while respecting node autonomy.*

3 The Query Trading Algorithm

3.1 Query Trading Overview

The idea of the basic query trading (QT) algorithm is to consider queries and query-answers as commodities and the query optimization procedure as a trading of query-answers between nodes holding information that is relevant to the contents of these queries. Buying nodes are those that are unable to answer some query, either because they lack the necessary resources (e.g. data, I/O, CPU), or simply because outsourcing the query is better than having it executed locally. Selling nodes are the ones offering to provide data relevant to some parts of these queries. Each node may play either role (buyer and seller) depending on the query been optimized and the data that each node locally holds.

Before proceeding with the presentation of the optimization algorithm, we should note that no query or part of it is physically executed during the whole optimization procedure. The buyer simply asks from seller for assistance in evaluating some *queries* and sellers make offers which contain their *estimated* properties of the answer of these queries (*query-answers*). These properties can be the total time required to execute and transmit the results of the query back to the buyer, the time required to find the first row of the answer, the average rate of retrieved rows per second, the total rows of the answer, the freshness of the data, the completeness of the data, and possibly a charged amount for this answer.

The buyer ranks the offers received using an administrator-defined weighting aggregation function and chooses those that minimize the total cost/value of the query. In the remaining of this section, the valuation of the offered query-answers will be the total execution time (cost) of the query, thus, we will use the terms cost and valuation interchangeably.

3.2 The Query-Trading Algorithm

The execution plans produced by the query-trading (QT) algorithm consist of the query-answers offered by remote sellers together with the processing operations

required to construct the results of the optimized queries from these offers. The task of the algorithm is to find the combination of data offers and buyer and seller processing operations that minimize the valuation (cost) of the final answer. For this reason, it runs iteratively, progressively selecting the best execution plan. In each iteration, the buyer 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.

Buyer-side algorithm	**Sellers-side algorithm**
B0. Initialization, set $Q = \{\{q, C\}\}$ B1. Make estimations of the values of the queries in set Q, using a trading strategy. B2. Request offers for the queries in set Q	S1. For each query q in set Q do the following: S2.1. Find sub-queries q_k of q that can be answered locally. S2.2. Estimate the cost c_k of each of these sub-queries q_k. S2.3. Find other (sub-)queries that may be of some help to the buyer.
B3. Select the best offers $\{q_i, c_i\}$ using **one of the three** methods (bidding, auction, bargaining) of the query trading framework	S3. Using the query trading framework, make offers and try to sell some of the subqueries of step S2.2 and S2.3.
B4. Using the best offers, find possible execution plans P_m and their estimated cost C_m B5. Find possible sub-queries q_e and their estimated cost c_e that, if available, could be used in step B4. B6. Update set Q with sub-queries $\{q_e, c_e\}$. B7. Let P_* be the best of the execution plans P_m. If P_* is better than that of the previous iteration of the algorithm, or if step B6 modified the set Q, then go to step B1. B8. Inform selling-nodes, which queries are used in the best execution plan P_*, so that they start executing these queries.	

Fig. 1. The query trading (QT) algorithm.

The optimization algorithm is actually a kind of bargaining between the buyer and the sellers. The buyer asks for certain queries and sellers counter-offer to evaluate some (modified parts) of these queries at different values. In each iteration of this bargaining the negotiated queries are different, as the buyer **and** sellers progressively identify additional queries that may help in the optimization procedure. This difference, in turn, makes necessary to change selling nodes in each step of the bargaining, as these additional queries may be better offered by other nodes.

Figure 1 presents the details of the distributed optimization algorithm. The input of the algorithm is a query q with an initially estimated cost of C. If no estimation using the available local information is possible, then C is a predefined constant (zero or something else depending on the type of cost used). The output is the estimated best execution plan P_* and its respective cost C_* (step B8). The algorithm, at the buyer-side, runs iteratively (steps B1 to B7). Each iteration starts with a set Q of pairs of queries and their estimated costs, which the buyer

would like to purchase from remote nodes. In the first step (B1), the buyer strategically estimates the values it should ask for the queries in set Q, and then asks for bids (RFB) from remote nodes (step B2). The (candidate) sellers after receiving this RFB make their offers, which contain query-answers concerning parts of the queries in set Q (step S2.1 - S2.2) or other relevant queries that could be of some use to the buyer (step S2.3). The winning offers are then selected using a small nested trading negotiation procedure (steps B3 and S3). The buyer uses the contents of the winning offers to find a set of candidate execution plans P_m and their respective estimated costs C_m (step B4), and an enhanced set Q of queries-costs pairs (q_e, c_e) (steps B5 and B6) which they could possibly be used in the next iteration of the algorithm for further improving the plans produced at step B4. Finally, in step B7, the best execution plan P_* out of the candidate plans P_m is selected. If P_* is not better than that produced in the previous iteration (i.e., no further improvement is possible) and step B5 did not find any new query, then the algorithm is terminated.

4 Query Allocation

Many query allocation mechanisms, (e.g., [1]), including ours, classify queries into a large number of disjoint classes, e.g., few 1000s. We assume a set Q of K query templates/classes, $Q = \{q_1, q_2, \ldots, q_K\}$, where each template represents a family of queries differing only in some selection constant(s) in their qualification. If a query can be derived from template q_k, it is a q_k-class query.

Let I be the number of nodes in the system and K be the number of different query classes. During a small time period τ with duration T, the behavior of each node i $(1 \leq i \leq I)$ can be completely captured using the query *demand*, *consumption*, and *supply* vectors.

The demand vector $\boldsymbol{d_i} = (d_{i1}, d_{i2}, \ldots, d_{iK}) \in N^K$ contains the number of queries $(q_1, q_2, \ldots, q_K)$ posed to node i during τ. The respective consumption vector $\boldsymbol{c_i} = (c_{i1}, c_{i2}, \ldots, c_{iK})$ contains the number of those queries that are actually evaluated by the system, either locally or at a distant node $(c_{ik} \leq d_{ik}, 1 \leq i \leq I, 1 \leq k \leq K)$. Finally, the supply vector $\boldsymbol{s_i} = (s_{i1}, s_{i2}, \ldots, s_{iK}) \in N^K$ contains the number of queries $(q_1, q_2, \ldots, q_K)$ evaluated by node i during τ (whether initiated at i or elsewhere). The set of all feasible supply vectors $\boldsymbol{s_i}$ of node i depends on its available hardware resources and is the *supply set* (S_i) of node i.

Given the nodes' demand vectors $\boldsymbol{d_i}$ for a time period τ and supply sets S_i, $(i = 1, 2, \ldots, I)$, a query allocation mechanism finds consumption and supply vectors that satisfy certain optimality criteria or other constraints. Such a solution is denoted as $<[\boldsymbol{s_i}], [\boldsymbol{c_i}]>$, where $\boldsymbol{s_i} \in S_i$, $\boldsymbol{c_i} \in N^K$, $1 \leq i \leq I$. Generally, if the distributed system is not overloaded, we expect from query allocation mechanisms to find solutions having $\boldsymbol{d_i} = \boldsymbol{c_i}$, $i = 1, 2, \ldots, I$, i.e., nodes getting answers for all queries within τ. Otherwise, some queries will be delayed and will be counted in the demand vectors of subsequent time periods as well.

In addition to individual nodes' vectors, we also use system-wide *aggregate* demand ($\boldsymbol{d}$), supply ($\boldsymbol{s}$), and consumption ($\boldsymbol{c}$) vectors defined as:

$$\boldsymbol{d} = \sum_{i=1}^{I} \boldsymbol{d_i}, \qquad \boldsymbol{s} = \sum_{i=1}^{I} \boldsymbol{s_i}, \qquad \boldsymbol{c} = \sum_{i=1}^{I} \boldsymbol{c_i} \tag{1}$$

In the same spirit, one may obtain the aggregate supply set S capturing the capabilities of all nodes of the system, by combining the individual supply sets of the nodes, each time summing up one supply vector from each node:

$$S = \{\boldsymbol{s} \in N^K : \boldsymbol{s} = \sum_{i=1}^{I} \boldsymbol{s_i}, \;\; \boldsymbol{s_i} \in S_i\} \tag{2}$$

Based on the semantics of aggregate vectors, at any time period τ, the aggregate query supply is equal to the aggregate query consumption, which is at most equal to the aggregate query demand:

$$\boldsymbol{s} = \boldsymbol{c} \leq \boldsymbol{d}, \;\; \boldsymbol{s} \in S \tag{3}$$

Sometimes, there is no feasible way for the system to evaluate all queries requested in a time period. In such cases, each node i selects its consumption vector based on a *preference relation* ($\preceq_i$) over the set of all possible such vectors. The semantics of $\preceq_i$ is that, if $\boldsymbol{c_i}, \boldsymbol{c_i}' \in N^K$ and $\boldsymbol{c_i} \succeq_i \boldsymbol{c_i}'$, then node i prefers the $\boldsymbol{c_i}$ query consumption vector over $\boldsymbol{c_i}'$. In the remainder of this thesis summary and without loss of generality, we assume that all nodes prefer to evaluate as many queries as possible, independent of what these queries are: $\boldsymbol{c_i} \succeq_i \boldsymbol{c_i}'$ iff $\sum_{k=1}^{K} c_{ik} \geq \sum_{k=1}^{K} c'_{ik}$. Using this preference relation, our algorithm will find solutions that maximize the number of queries evaluated in each time period. The role of preference relations in optimizing the choice of consumption vectors by query allocation mechanisms is formalized through the notion of *Pareto optimality.*

Definition 1 (Pareto Optimality). *A solution* $<[\boldsymbol{s_i}], [\boldsymbol{c_i}]>$ $(1 \leq i \leq I)$ Pareto dominates *a solution* $<[\boldsymbol{s_i}'], [\boldsymbol{c_i}']>$ *iff*

$$\begin{array}{ll} \forall\, 1 \leq i \leq I \quad : \boldsymbol{c_i} \succeq_i \boldsymbol{c_i}', & \textit{and} \\ \exists\, g, 1 \leq g \leq I : \boldsymbol{c_g} \succ_g \boldsymbol{c_g}' & \end{array}$$

That is, all nodes prefer their consumption vector $\boldsymbol{c_i}$ *to* $\boldsymbol{c_i}'$ *(*$\boldsymbol{c_i} \succeq_i \boldsymbol{c_i}'$*) and at least one of them (i.e., node g) strictly prefers* $\boldsymbol{c_g}$ *to* $\boldsymbol{c_g}'$ *(*$\boldsymbol{c_g} \succ_g \boldsymbol{c_g}'$*). A solution is* Pareto optimal *if it is not Pareto dominated by any other solution.*

Since node preferences maximize the number of queries evaluated per time period, *a Pareto optimal allocation is one that no node can further increase the number of queries consumed without reducing those of another node.*

Based on all the above, the problem presented in the introduction of this chapter is formally stated as follows:

Problem 1 (Query Allocation (QA)). Given a federation of autonomous database systems with supply sets S_i, preference relations $\succeq_i$ $(i = 1, 2, \ldots, I)$, and for a time period τ of length T, query demand vectors $\boldsymbol{d_i}$ $(i = 1, 2, \ldots, I)$, Query Allocation (QA) seeks to find a Pareto optimal solution $<[\boldsymbol{s_i}], [\boldsymbol{c_i}]>$, $i = 1, 2, \ldots I$, for τ.

The goal of our work is to solve the QA problem in a completely *decentralized and autonomous* way. The feasibility of this attempt steams from the *First Theorem of Welfare Economics* (FTWE) [3]. According to FTWE, *market economies composed of self-interested consumers and firms achieve allocations of resources and goods that are Pareto optimal*. Moreover, the behavior of consumers and firms is such as if an *invisible hand* is guiding their actions toward a state beneficial to all.

Table 1. Mapping between microeconomics theory entities and entities of the QA problem.

Microeconomics		QA problem
Commodities markets		Query processing framework
Commodities		Queries
Buyers	$\Longleftrightarrow$	Client nodes
Sellers (Firms)		Server nodes
Commodity value: Monetary units		Query value: Virtual monetary units

Mapping Between QA and Microeconomics

We have already discussed the mapping between client/servers of the QA problem and buyer/sellers of the respective Microeconomics problem. A central construct of all competitive markets is that commodities have values(prices) measured using a monetary unit. This is a microeconomics mechanism designating the importance of each piece of commodity to the society. In the QA problem there is no such mechanism, therefore, we use a virtual monetary unit and assign a virtual value $p_k \in R_+$ $(1 \leq k \leq K)$ to each q_k query. The resulting virtual query prices are only used by our solution and are otherwise useless.

If we use vector notation, then the price vector $\boldsymbol{p} = (p_1, p_2, \ldots, p_K) \in R_+^K$ will describe the (virtual) value of a unit of each of the K query classes. The value of a consumption vector $\boldsymbol{c_i}$ can be calculated as $\sum_{k=1}^{K} p_k c_{ik}$, which is written in vector notation as $\boldsymbol{p} \cdot \boldsymbol{c_i}$. Similarly, the value of a supply vector of seller i is $\boldsymbol{p} \cdot \boldsymbol{s_i}$.

Table 1 summarizes the way we mapped the entities of the QA problem to microeconomics.

4.1 Query Market Definition

What remains for FTWE to hold is to make nodes act as if they participate in traditional competitive commodity markets. We do so in this section and show

that this behavior implicitly leads clients and servers to make Pareto optimal allocations of queries to nodes.

In competitive markets, each seller is assumed selfish and selects to supply the vector $\boldsymbol{s_i^\star} \in S_i$ with the largest (virtual) value. That is, sellers/servers solve the following problem:

$$\boldsymbol{p} \cdot \boldsymbol{s_i}^\star = \max_{\boldsymbol{s_i} \in S_i} (\boldsymbol{p} \cdot \boldsymbol{s_i}) \qquad i = 1, 2, \ldots, I \tag{4}$$

In general commodities markets, the purchasing power of buyers (i.e., client nodes in our problem) is limited by their wealth. In our case, we want to maximize the number of queries evaluated per time period. Therefore, we put no consumption limit to nodes, apart from the fact that the resulting aggregate supply ($\boldsymbol{s^\star} = \sum_{i=1}^{I} \boldsymbol{s_i^\star}$) and consumption ($\boldsymbol{c^\star} = \sum_{i=1}^{I} \boldsymbol{c_i^\star}$) vectors should be equal.

If we choose a random price vector $\boldsymbol{p}$ and solve equation (4) we end up with demand and supply vectors that do not satisfy (3). This is captured in microeconomics using the notion of excess demand defined below:

Definition 2 (Excess demand). *Given prices* $\boldsymbol{p}$*, the* excess demand $z_k(\boldsymbol{p})$ *for* q_k*-queries is given by*

$$z_k(\boldsymbol{p}) = \sum_{i=1}^{I} d_{ik} - s_{ik} \tag{5}$$

The excess demand of all query classes will be denoted by the vector

$$\boldsymbol{z}(\boldsymbol{p}) = (z_1(\boldsymbol{p}), z_2(\boldsymbol{p}), \ldots, z_K(\boldsymbol{p}))$$

The sign of the excess demand $z_k(\boldsymbol{p})$ for q_k reveals whether they supply of q_k by server (seller) nodes is larger ($z_k(\boldsymbol{p}) < 0$) or smaller ($z_k(\boldsymbol{p}) > 0$) than what the current client (buyer) workload demands.

We can now formally define the term *market equilibrium* that was first mentioned in FTWE.

Definition 3 (Market competitive equilibrium). *A market is in a competitive equilibrium iff commodities prices* $\boldsymbol{p^\star}$ *are such that* $z(\boldsymbol{p^\star}) = 0$.

FTWE asserts that in equilibrium the resulting distribution of queries is Pareto optimal. Thus, if we calculate the equilibrium price vector $\boldsymbol{p}^\star$, the resulting virtual query market will solve the QA problem. This is shown in the next section.

4.2 The Pricing Mechanism

Traditionally, microeconomic theory finds equilibrium prices using a *tâtonnement* process (TP) which iteratively adjusts prices until the excess demand is zero for all commodities. It assumes that there is a single entity called *umpire* that has the role of market coordinator. It iteratively announces to all entities a single market price (per commodity), collects their consumption and supply vectors

for these commodities, adjusts prices, and then a new iteration is started by announcing the new prices. The iteration is stopped when consumption equals supply. It is possible to modify the tâtonnement process in such a way that no centralized authority is required and trading takes place before equilibrium is reached. Examples of such modifications are given in [4].

QA-NT: Non-tâtonnement price adjustment algorithm (runs at each server node i)

1 Repeat for ever
2 Given the current prices $\boldsymbol{p}$ of queries, solve (4) (first order conditions). This will calculate the optimal supply vector $\boldsymbol{s_i} \in N^K$ of the node.
3 While a time period τ has not elapsed do.
4 If a client node asks to evaluate a query q_k and $s_{ik} > 0$ then
5 Offer to evaluate the query.
6 If offer is accepted set $s_{ik} = s_{ik} - 1$.
7 Else
8 Do not offer to evaluate query q_k.
9 Set $p_k = p_k + \lambda p_k$.
10 End If
11 End while
12 For each k s.t. $s_{ik} > 0$ do
13 Set $p_k = p_k - s_{ik} \lambda p_k$
14 End For
15 End Repeat

Fig. 2. The QA-NT algorithm

Figure 2 presents the our query alloocation algorithm (QA-NT). This is based on a non-tâtonnement process where no centralized authority is needed. Query prices are never disclosed or exchanged over the network. Each node calculates its own set of prices and uses them only to calculate its own supply vector (step 2 of the QA-NT algorithm). Thus, there is no need for all nodes to use the same K query classes, which is difficult to calculate in a decentralized way. The only restriction is that for each node, queries belonging to the same query class should require similar resources for their evaluation on that node.

Step 4 of the non-tâtonnement algorithm describes the negotiation strategy of servers, i.e., they do not try to be fair and immediately accept a request to evaluate query q_k iff $s_{ik} > 0$. If all available servers reject a request for a query, the respective client resubmits it in the next time period.

Proposition 1. *If the non-tâtonnement algorithm is left running for a long time period, then* $\lim_{t \to \infty} z(\boldsymbol{p}) = \boldsymbol{0}$.

Proof. The proof is quite complicated and is given for the general case of non-tâtonnement processes in [4].

5 Conclusion

In this thesis we discuss a novel query optimization and allocation algorithm suitable for networks of autonomous data management systems. Our algorithms are inspired by Microeconomics.

References

1. P. E. Drenick and E. J. Smith. Stochastic query optimization in distributed databases. *ACM Trans. Database Syst.*, 18(2):262–288, 1993. 4
2. Donald Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys*, 34(4):422–469, September 2000. 1
3. Andreu Mas-Colell, Michael D. Whinston, and Jerry R. Green. *Microeconomic Theory*. Oxford University Press, New York, 1995. 4
4. Anjan Mukherji. Competitive equilibria: Convergence, cycles or chaos, The seventh Int. meeting of the society for social choice and welfare, discussion papers. Technical report, Institute of Social and Economic Research, Osaka University, Japan, July 2003. 4.2, 4.2
5. Fragkiskos Pentaris and Iannis Ioannidis. Autonomic query allocation based on microeconomics principles. In Rada Chirkova, Asuman Dogac, Tamer Ozsu, and Timos Sellis, editors, *Proceedings of the 2007 IEEE 23rd International Conference onData Engineering, April 15-20, 2007, Istanbul, Turkey*. IEEE Computer Society, 2007. ⋆
6. Fragkiskos Pentaris and Yannis E. Ioannidis. Distributed query optimization by query trading. In Elisa Bertino, Stavros Christodoulakis, Dimitris Plexousakis, Vassilis Christophides, Manolis Koubarakis, Klemens Böhm, and Elena Ferrari, editors, *EDBT*, volume 2992 of *Lecture Notes in Computer Science*, pages 532–550. Springer, 2004. 1, ⋆
7. Fragkiskos Pentaris and Yannis E. Ioannidis. Query optimization in distributed networks of autonomous database systems. *ACM Transactions on Database Systems*, 31(2):537 – 583, June 2006. 1, ⋆
8. 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. 1