Autonomic Query Allocation based on Microeconomics Principles [*]

Fragkiskos Pentaris and Yannis Ioannidis
Department of Informatics and Telecommunications, University of Athens,
Panepistemioupolis, 15771, Athens, Greece,
E-mail:{frank,yannis}@di.uoa.gr

Abstract

In large federations of autonomous database systems, automatic distribution of the query workload to those systems is a critical issue. We examine this problem under the perspective of microeconomics theory and show how the latter can be used to construct an efficient decentralized mechanism that maximizes system throughput. In particular, we introduce a solution that is based on the notion of query markets. We examine the properties of these markets and show that they result in Pareto-optimal allocations of resources to queries. An extensive set of experiments with both a simulator and an actual implementation on top of a commercial DBMS demonstrate significant improvements in the overall system throughput when our technique is used.

1 Introduction

Several emerging applications depend on distributed systems that loosely integrate autonomous data management servers. Examples include federated DBMSs operating in intranets of large companies as well as GRID-based systems. Since the workload of nodes may exhibit large fluctuations, their administrators spend much of their time with query monitoring tools, designing workload-distribution policies that avoid overloading of critical nodes. Unfortunately, the effectiveness of these policies is hard to predict as they usually assume a static workload distribution.

Typically, a large distributed system follows a multi-way data mirroring policy to reach the scalability and availability levels required by its users. The hardware used is selected in such a way that under normal conditions, peek load does not exceed a threshold that is safely below total system capacity. There are always cases, however, where load temporarily exceeds this limit or even total system capacity itself. This can be due, for example, to multiple node failures or even to singularities of the business logic of the system. Using an automated, self-tuning, and quickly reacting query allocation mechanism that maximizes system query throughput during these conditions [3] is very important. It ensures that system capacity stays as high as possible and, in case of temporary overloads, the duration of resource unavailability is minimized.

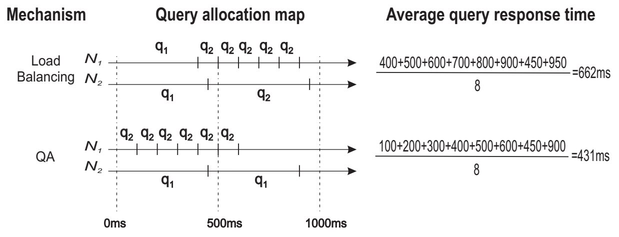

Mechanism
Query allocation map
Average query response time
Load Balancing
N_1
N_2
q_1
q_2
QA
$\frac{400+500+600+700+800+900+450+950}{8}$=662ms
$\frac{100+200+300+400+500+600+450+900}{8}$=431ms
0ms
500ms
1000ms

Figure 1: Performance optimization vs Load Balancing.

Existing query allocation mechanisms (e.g., [1,2,4,8,10] balance the load of all system nodes hoping that, as a side-effect, system performance will also be optimized. Unfortunately, this is not always true. For instance, consider a system with only two nodes, N_1 and N_2, having a workload consisting of queries q_1 and q_2. Node N_1 evaluates q_1 and q_2 in 400ms and 100ms, respectively, whereas N_2 does so in 450ms and 500ms, respectively. For simplicity, assume that no node can evaluate two queries simultaneously. The system uses a typical greedy load balancing (LB) algorithm that assigns each incoming query to the node that would result in the least load imbalance among all nodes. Supposed now that, in a very short time period, different applications running at node N_1 ask from the distributed system to evaluate one q_1 and six q_2 queries. Similarly, node N_2 asks for the evaluation of one q_1. For simplicity, let requests for q_1 arrive to the system before those for q_2. The LB algorithm will assign the first two q_1 queries to nodes N_1 and N_2 to reduce load imbalance. Subsequently, it will assign the first three q_2 queries to N_1, the fourth q_2 to N_2,

[*]This research was partially supported by the Information Society Technologies (IST) Program of the European Commission under the DELOS Network of Excellence on Digital Libraries (Contract G038-507618) and the BRICKS Integrated Project (Contract 507457).

and finally, the last two q_2 to N_1. Figure 1 shows that with this assignment, N_1 and N_2 will be busy processing queries for 900ms and 950ms, respectively (50ms total load imbalance) and the average query response time experienced by the applications will be 662ms per query.

To minimize query response time, there is actually a much better query allocation strategy, which is labeled QA in Figure 1. QA has nodes N_1 and N_2 accept only q_2 and q_1 queries, respectively. The average query response time in this case is 431ms. Note that, not only LB is 54% slower than QA, but also dangerously prolongs the overload period (i.e., the period where all nodes are busy) by 50%, as QA leaves node N_1 idle after 600ms whereas LB does so after 900ms.

In this paper, we present a *self-tuning, completely decentralized, and dynamic query allocation mechanism* suitable for autonomous federations of DBMSs. It is inspired by microeconomics and maintains its efficiency even during load fluctuations. To the best of our knowledge, it is the first algorithm of its kind, fully respecting the autonomy of participating systems. Furthermore, its novelty lies also with the fact that it is allocating queries not by equalizing the load of nodes, but by optimizing system performance instead.

The remainder of the paper is organized as follows: In Section 2, the query allocation problem is formally presented. In Section 3, we present our solution. In Section 4, we discuss the differences of our approach to other existing ones, and in Section 5, we experimentally evaluate our algorithm.

2. Description of the Problem

2.1. Execution Environment

Our execution environment is a disparate network of autonomous relational DBMSs, which may act in a cooperative or a competitive manner. All nodes are treated as black boxes that externally understand a common relational data schema. Each of them may have different processing capabilities and a different set of locally held relations or partitions of them. Nodes may act as servers, evaluating queries on behalf of other nodes, clients, asking from other nodes to evaluate certain queries, or both.

During query processing, a node communicates with other nodes that may have necessary data or resources that the original node is missing and, subsequently, assigns query evaluation tasks to the nodes that have offered the best solutions for the corresponding query parts (under some definition of optimality). Examples of such query optimization mechanisms are MARIPOSA [15] and the QT and SQPT algorithms in the recently proposed Query and Process Trading framework [13, 14]. Describing in detail such mechanisms is beyond the scope of this paper.

Although our mechanism is independent of the type of queries and the data model used, for readability reasons and without loss of generality, we assume that the workload consists of read-only, SQL-like select-join-project-sort queries. Many query allocation mechanisms, 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. These constants are such that, queries of the same template use similar resources and have similar estimated execution cost *when run on the same node* (could be different on different nodes). If a query can be derived from template q_k, it is a q_k-class query.

In a disparate and dynamic environment, identification of set Q is difficult and requires pieces of information that compromise node autonomy. As we show later, our algorithm allows each node to proceed with its own private classification of queries, without harming node autonomy. Nevertheless, for readability, our presentation assumes that all nodes identify the same query-class set Q.

2.2. Problem Modeling

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 $\vec{d_i} = (d_{i1}, d_{i2}, \ldots, d_{iK}) \in \mathbb{N}^K$ contains the number of queries $(q_1, q_2, \ldots, q_K)$ posed to node i during τ. The respective consumption vector $\vec{c_i} = (c_{i1}, c_{i2}, \ldots, c_{iK})$ 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 $\vec{s_i} = (s_{i1}, s_{i2}, \ldots, s_{iK}) \in \mathbb{N}^K$ 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 $\vec{s_i}$ of node i depends on its available hardware resources and is the *supply set* (S_i) of node i.

For instance, in our earlier example, $I = K = 2$. Assuming that $T = 500$ms, then in the first time period τ (0ms-499ms), the demand vector of node N_1 was $\vec{d_1} = (1, 6)$, and using the LB mechanism, its consumption vector was $\vec{c_1} = (1, 1)$. That is, N_1 asked from the distributed system to evaluate one q_1 and six q_2 queries, but only one q_1 and one q_2 of those were actually evaluated. Finally, the respective supply vector of N_1 was also $\vec{s_1} = (1, 1)$, since in that time period it evaluated one q_1 and one q_2 queries.

Given the nodes' demand vectors $\vec{d_i}$ 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

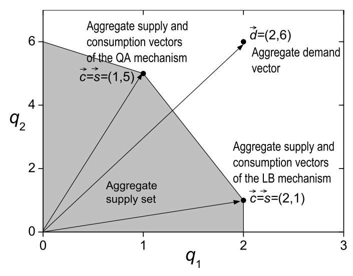


Figure 2: The aggregate demand, supply and consumptions vectors.

solution is denoted as $<[\vec{s_i}], [\vec{c_i}]>$, where $\vec{s_i} \in S_i, \vec{c_i} \in \mathbb{N}^K$, $1 \leq i \leq I$. Generally, if the distributed system is not overloaded, we expect from query allocation mechanisms to find solutions having $\vec{d_i} = \vec{c_i}, i = 1, 2, \ldots, I$, 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 ($\vec{d}$), supply ($\vec{s}$), and consumption ($\vec{c}$) vectors defined as:

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

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:

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

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:

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

Figure 2 refers to our earlier example system and its first time period, and shows the aggregate query demand vector $\vec{d}$ and the aggregate consumption and supply vectors $\vec{c}$ and $\vec{s}$, for both the LB and the QA strategies. The gray area represents the aggregate supply set S of the system.

Note that the aggregate demand vector $\vec{d}$ is outside the gray area, implying that there was no feasible way for the system to evaluate all queries requested in the first 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, i.e., there is some type of negotiation among nodes. The semantics of $\preceq_i$ is that, if $\vec{c_i}, \vec{c_i}' \in \mathbb{N}^K$ and $\vec{c_i} \succeq_i \vec{c_i}'$, then node i prefers the $\vec{c_i}$ query consumption vector over $\vec{c_i}'$.

In the remainder of this paper and without loss of generality, we assume that all nodes prefer to evaluate as many queries as possible, independent of what these queries are: $\vec{c_i} \succeq_i \vec{c_i}'$ 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.

Continuing with our earlier example and the first time period, Figure 1 shows that both LB and QA used the same resources. However, according to Figure 2, LB does not follow the nodes' preference relation: the difference between the demand vector $\vec{d}$ and the aggregate supply vector of LB is larger than that of QA.

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

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

That is, all nodes prefer their consumption vector $\vec{c_i}$ *to* $\vec{c_i}'$ ($\vec{c_i} \succeq_i \vec{c_i}'$) *and at least one of them (i.e., node* g*) strictly prefers* $\vec{c_g}$ *to* $\vec{c_g}'$ ($\vec{c_g} \succ_g \vec{c_g}'$)*. 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.* For instance, in our case example, the LB solution was not Pareto, as nodes N_1 and N_2 consumed 2 and 1 queries, respectively, which is Pareto dominated by the QA allocation that had nodes N_1 and N_2 consume 5 and 1 queries, respectively. Note that the definition of Pareto optimality involves only the consumption vectors of nodes. Their supply vectors are involved indirectly through equations (1) and (3). In general, more than one Pareto optimal solution may exist.

Based on all the above, the problem presented in the introduction of this paper 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* $\vec{d_i}$ $(i = 1, 2, \ldots, I)$*, Query Allocation (QA) seeks to find a Pareto optimal solution* $<[\vec{s_i}], [\vec{c_i}]>$*,* $i = 1, 2, \ldots I$*, for* τ*.*

The goal of our work is to solve the QA problem in a completely *decentralized and autonomous* way. Our solution to this challenging task is given in the next section.

3. The Query Markets

The main idea of this paper is to use microeconomics theory to find Pareto optimal query allocations in a completely distributed way. The feasibility of this attempt steams from the *First Theorem of Welfare Economics* (FTWE) [9]. 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.

To use microeconomics theory for Query Allocation, we must do the following:

(i) map the entities of the QA problem (e.g., server and client nodes) to those used in traditional microeconomics (e.g., sellers and buyers, respectively);
(ii) define a competitive market using the mapped entities;
(iii) use this market to solve the problem in microeconomics that is equivalent to QA.

We discuss each of these steps separately below.

3.1. Mapping Between QA and Microeconomics

The idea binding the QA problem and FTWE is that we consider queries as the traded commodities. Query processing is then modeled as a task of query trading between nodes holding information relevant to the contents of the queries being evaluated. Buying nodes (consumers) are those that are unable to answer some query, either because they lack the necessary resources (e.g., relevant data or CPU cycles), or simply because outsourcing the query is better than executing it locally. Selling nodes (firms) are those having data relevant to some parts of these queries or having excess resources. Each node may play any of those two roles (buyer and seller) depending on the query being evaluated and the data it locally holds. Thus, the role of servers and clients in the QA problem are played by sellers/firms and buyers/consumers in the competitive market, respectively.

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

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

3.2. 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 $\vec{s_i^\star} \in S_i$ with the largest (virtual) value. That is, sellers/servers solve the following problem:

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

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 ($\vec{s^\star} = \sum_{i=1}^{I} \vec{s_i^\star}$) and consumption ($\vec{c^\star} = \sum_{i=1}^{I} \vec{c_i^\star}$) vectors should be equal.

If we choose a random price vector $\vec{p}$ 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 $\vec{p}$, the* excess demand *$z_k(\vec{p})$ for q_k-queries is given by*

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

The excess demand of all query classes will be denoted by the vector $\vec{z}(\vec{p}) = (z_1(\vec{p}), z_2(\vec{p}), \ldots, z_K(\vec{p}))$.

The sign of the excess demand $z_k(\vec{p})$ for q_k reveals whether they supply of q_k by server (seller) nodes is larger ($z_k(\vec{p}) < 0$) or smaller ($z_k(\vec{p}) > 0$) 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 $\vec{p^\star}$ are such that $z(\vec{p^\star}) = 0$.*

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

3.3. 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

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

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

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. The iterative price adjustment process is given by

$$\vec{p}^{(t+1)} = \vec{p}^{(t)} + \lambda z(\vec{p}^{(t)}) \tag{6}$$

where $\vec{p}^{(t+1)}$ is the new price vector at time $(t+1)$ given prices $\vec{p}^{(t)}$ and excess demand $z(\vec{p}^{(t)})$ caused by these prices $\vec{p}(t)$. The adjustment process increases the prices of queries that are in excess demand (i.e., $z_j(\vec{p}^{(t)}) > 0$) and reduces the prices of those that are excess supplied (i.e., $z_j(\vec{p}^{(t)}) < 0$). This indirectly causes nodes to increase supply of the former queries and reduce the supply of the latter. The value of parameter $\lambda \in \mathbb{R}_+^\star$ affects the number of iterations required by $\vec{p}^{(t)}$ to converge to equilibrium prices $\vec{p^\star}$. Higher values reduce the number of iterations but decrease the accuracy of the estimated vector $\vec{p^\star}$.

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 [9, 11, 12]. Trading between two nodes in disequilibrium (i.e., non-equilibrium) prices is allowed according to the following rule:

Definition 4 (Non-tâtonnement trading rule) *Let $c_i^{(t)}$ and $s_j^{(t)}$ denote the consumption and supply vector of node i and j at time t, respectively. We allow nodes i and j to increase their consumption and supply vectors by $\vec{d}$, respectively, even in disequilibrium prices, iff the following hold*

1. *the new supply vector $\vec{s_j}^{(t+1)} = \vec{s_j}^{(t)} + \vec{d}$ of node j is an element of its supply set, i.e., it is feasible.*

2. *exhausts all possibilities of other trade. That is*

$$\vec{c_i}^{(t+1)} \succeq_i \vec{c_i}^{(t)} + \vec{\epsilon}$$

for all $\vec{\epsilon} \in \mathbb{N}^K$ s.t. the supply vector $\vec{s_j}^{(t)} + \vec{\epsilon}$ belongs to the supply set of node j, respectively.

Rule 1 ensures that the trading is feasible. Rule 2 ensures that the non-tâtonnement process converges to a unique equilibrium. Any buyer (seller) that does not manage to consume (supply) the queries it wants to acquire (produce) immediately infers that prices are not in equilibrium and adjusts its own prices up (down) proportionally to the quantity that was not consumed (supplied). These trading failures are the only reason for adjusting prices. The following *decentralized algorithm (QA-NT)* describes the non-tâtonnement process:

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

```
1  Repeat for ever
2    Given the current prices $\vec{p}$ of queries, solve (4) (first
     order conditions). This will calculate the optimal
     supply vector $\vec{s_i} \in \mathbb{N}^K$ of the node.
3    While a time period $\tau$ 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
```

The description of the non-tâtonnement algorithm shows that 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 3.1 *If the non-tâtonnement algorithm is left running for a long time period, then* $\lim_{t\to\infty} z(\vec{p}) = \vec{0}$.

Proof 1 *The proof is quite complicated and is given for the general case of non-tâtonnement processes in [11].*

To make more concrete to the reader how the query market solution distributes query workload among nodes, consider again our test case example and assume that equilib-

Mechanism	Distributed	Workload type	Conflict with query optimization	Autonomy	Performance
QA-NT	X	Dynamic	-	X	Very Good
Greedy	X	Dynamic	X	-	Very Good
Random	X	Dynamic	X	X	Poor
Round-robin	X	Dynamic	X	X	Poor
BNQRD	X	Dynamic	X	-	Poor
Markov	-	Static	X	-	Excellent

Table 2: Comparison of query allocation mechanisms

rium prices are initially $\vec{p^\star} = (1, 1)$. By solving (4), node N_1 will supply only q_2 queries. Assume now that query distribution is modified and demand for queries q_1 cannot be satisfied (by node N_2). Then, prices of q_1 queries will start increasing until node N_1 starts to also supply q_1. The actual query optimization and processing mechanisms of the system do not have to use any economics-based ideas or the fictitious prices of queries. We simply let an economy run in parallel with the actual query processing mechanisms. The only role of the query economy is to calculate the supply of queries from nodes at each time period. That is, our mechanism is a kind of query admission control.

Before going on to discuss existing work on query allocation mechanisms it is worth to briefly mention that the mapping between the query allocation problem and query markets is such that all requirements of the *Second Theorem of Welfare Economics (STWE)* [9] are satisfied. This means that *any Pareto optimal solution produced by any non-microeconomics based query allocation algorithm can also be calculated using a modified version of our algorithm*, that is, our approach is very general. The modifications required are not discussed in this paper due to paper-size constraints but can be found in standard microeconomics textbooks such as [7].

4. Related Work

Automatic load balancing has recently attracted a lot of attention due to spreading of grid-based and cluster-based distributed databases. For instance, a leading commercial DBMS provider has recently implemented a client-level load balancing mechanism for its cluster solution. Clients use either a *random* or a *round-robin* strategy. That is, they attempt to balance the servers' load by randomly selecting the servers that will run their transactions, or simply by choosing servers in a round-robin mechanism. These approaches work well in homogeneous distributed databases (i.e., all nodes have the same schema and similar resources), but as we show in the experiments section, perform poorly in heterogeneous environments.

Another heuristic approach, frequently used for its simplicity, is to distribute queries in a *greedy* manner, i.e., immediately assign queries to server nodes that can evaluate them in the least time. A small amount of randomization may also be used to further improve performance. The greedy algorithm is easy to implement and performs surprisingly well, yet, it violates server node administrative autonomy, as clients unilaterally assign queries to servers.

In [4], a stochastic mechanism based on Markov chains and queueing theory is described that enables nodes to optimally assign queries to nodes. This mechanisms has excellent performance and produces Pareto optimal solutions, yet it suffers from scalability problems (it is a centralized mechanism). Furthermore, it assumes that query execution times are constant and workload is *static*, which is a major drawback. Finally, it is not compatible with autonomous systems due to pieces of information needed from nodes, like for instance the node capabilities.

In [1] and [2] the BNQRD algorithm for load balancing a locally distributed database system is examined. This algorithm uses a centrally calculated *unbalance factor* for each network node and assigns queries in such a way that CPU and I/O usage is evenly spread over the network. The BNQRD algorithm does not respect node autonomy since it requires from nodes to disclose information concerning their current load. Furthermore, BNQRD does not produce Pareto optimal results and thus, as we show in the experiments section, is inferior to our solution.

In [10], a simple load balancing technique that probes two eligible servers at random and chooses the one that has the least current load is presented. This *two-random probes* technique requires very few network messages and shows the advantages of using some randomness to prevent queries from overloading certain servers of a system. Still, the performance of this algorithm is far from optimal.

Although QA-NT requires more network messages than the previously mentioned algorithms, it usually outperforms all of them under dynamic loads and comes close to the Markov-based algorithm under static ones. This is because it treats each class of queries separately. Furthermore, it does not harm node autonomy by requiring very few pieces of information to properly work. In fact, it is the only one that truly respects administrative node autonomy by letting nodes decide for themselves the queries they will (offer to) evaluate. It can even work without problems in cases where only a subset of the nodes is using QA-NT, in which case it will still optimize global system throughput by modifying the behavior of only those nodes. All other previously mentioned algorithms will optimize the throughput of only the part of the system that uses them.

Complementary to query allocation is the effort in [5] and [6] on a schema optimization mechanism for DBMSs based on microeconomic theory. This mechanism improves query performance by distributing, *a priori*, the database tables in a Pareto optimal way. That is, the mechanism works by slowly modifying node schemas so that future query workloads can be better distributed by an optimal query allocation mechanism. Our algorithm QA-NT uses microeconomic theory as well but solves a completely different problem: having fixed the node schemas, it optimizes query workload distribution. In principle, QA-NT can work in parallel with any schema optimization mechanism, including that presented in [6].

Another related area of work is distributed query optimization for autonomous systems, where algorithms such as Mariposa [15] and Simultaneous Query and Process Trading (SQPT) [13, 14] split queries into pieces (subqueries) and then, assign these pieces to nodes so that queries response time is minimized. MARIPOSA [15] uses an economics-based two-phase mechanism for both query optimization and data distribution, whereas SQPT uses e-commerce techniques to split and assign both queries and processing to nodes. However, the corresponding pricing dynamics of the query markets used and the other properties of their pricing schemes have not yet been examined for either of the two approaches (to the best of our knowledge in the case of Mariposa). The QA-NT algorithm fills this gap and restricts the number of nodes offering to evaluate certain (sub)queries in a matter compatible with these algorithms. This is in contrast to other algorithms, which physically select a single node for each query, and thus, conflict with or render completely useless the existing distributed query optimization algorithms.

Table 2 summarizes our discussion on query allocation algorithms.

5. Experiment Study

In order to benchmark our algorithm (QA-NT) we implemented it over both a large simulated distributed network of DBMSs and a small one running the latest version of a leading commercial RDBMS. Testing the algorithm in a simulation was the only way to ensure its scalability in large networks, whereas testing it in a real (but inevitably small) network enabled us to see how QA-NT cooperates with real DBMSs.

5.1. Simulation

Experiments setup

Using C++, we implemented from scratch a simulator of a large federation of 100 autonomous RDBMSs. We implemented all algorithms presented in Section 4 except for the Markov-based one, since the latter cannot handle dynamic workloads. We run experiments with both homogeneous nodes (i.e., nodes with common local schema and capabilities) and heterogeneous ones (i.e., nodes having different

Parameter type	**Parameter**	**Value**
Network	Total size of Network	100 nodes
RDBMSs	Join capabilities	Merge-scan: All nodes. Hash-join:95 nodes
	CPU resources	One CPU 1-3.5GHz (2.3GHz avg.)
	Sorting/Hashing buffer size	2-10Mb per query per node (6Mbytes avg.)
	I/O Speed per node	5-80Mb/s (42.5Mb/s avg.)
Dataset	# of different relations	1,000
	Size of relations	1-20 Mbytes, (10.5Mbytes avg.)
	# of attributes per relation	10 attributes
	# of mirrors per relation	5 (avg.)
Workload	Joins per query	0 - 49 (24 average)
	Queries inter-arrival time	10-20000 ms (Zipf distribution)
	Average best execution time of queries	2000 ms
	Number of query classes	100
	Number of queries	10,000

Table 3: Simulation parameters.

schemas and capabilities). In the former case, all algorithms tested performed similar, therefore, we discuss only the results concerning heterogeneous environments. These are the most difficult to optimize, yet the most common ones in autonomous systems.

The parameters used in the experiments are displayed in Table 3. The dataset was synthetically created and consisted of 1,000 different relations with a size of 1-20Mbytes (avg. 10.5Mbytes). Each relation had 5 mirrors, one average, that where distributed randomly over the 100 RDBMSs. Each node had approximately 50 different relations.

The performance of QA-NT depends on the length T of each time period. Larger values of T increase QA-NT performance in static load but harms its flexibility with dynamic ones. In the experiments presented, T was set to 500ms. In each time period, we measured the number of queries executed and the average query response time of the algorithms. The latter was normalized by dividing it with the respective response time of QA-NT.

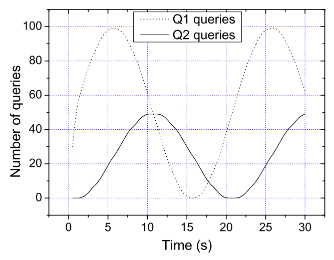


Figure 3: Example of sinusoid workload.

We run two sets of experiments. The first one examined QA-NT with regard to workload dynamics. We used a workload consisting of only two queries, Q1 and Q2, with an average execution time of 1000ms and 500ms, respectively (We have tried several other values for these as well

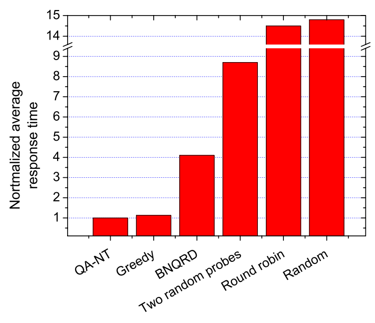


Figure 4: Average normalized query response time of the algorithms tested.

as other parameters but the nature of the results remain the same). We used two different queries, Q1 and Q2, to avoid trivial solutions. They were selected in such a manner that Q1 could be evaluated by all nodes, whereas Q2 could be evaluated by only half of the available nodes (i.e., only half of the nodes had the data required to answer them). The arrival rate of Q1 and Q2 always followed a sinusoid waveform with a 90^0 degrees phase difference between Q1 and Q2. The peek arrival rate of Q1 was always twice that of Q2. An example of such a workload is shown in Figure 3. The horizontal axis is the experiment time and the vertical one is the number of queries entering the distributed system per half second. Note that we did not use *impulse* loads (i.e., large loads of very small duration) or *unit-step* shaped ones, as these cannot be used to measure the behavior of non-linear algorithms like the ones tested in this Section.

The second set of experiments measured the behavior of our algorithm under a zipf-distributed workload. We synthetically created a workload having 10,000 queries belonging to 100 different query classes (select-join-project-sort queries having 0 to 49 joins). The average execution cost of these queries, when executed at the fastest (simulated) RDBMS without any other workload was approximately 2,000ms. The inter-arrival time of queries belonging to the same query class followed a zipf distribution with parameter $a = 1$. The maximum inter-arrival time between two queries was constraint to 30,000ms and the average inter-arrival time (t) was varied from 10ms to 20,000ms. Note that smaller query inter-arrival times mean larger rate of incoming queries and thus increased workload.

The Results

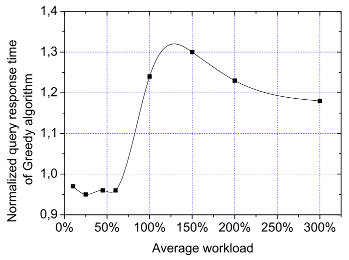
Sinusoids Workloads Initially we run some experiments using a 0.05Hz sinusoid load. Peek load was slightly bellow total system capacity. Figure 4 presents the normalized query response time of all algorithms tested. The QA-NT and the Greedy algorithms performed substantially better than the load balancing ones. The random and round-robing algorithms had the worst performance as they assigned equal amounts of queries to all nodes. Since nodes had different capacities, the resulting query distribution was far from optimal. The BNQRD algorithm balanced system load but still performed poorly, as it equalized the load of both the fast and the slow nodes. The QA-NT and the Greedy algorithms avoided doing so and assigned queries to slow nodes only when this maximized system throughput. Finally, the two-random probes algorithm performed better than the round-robin one but still, failed to completely balance the load across nodes, and thus, performed worse than BNQRD.

The behavior of our approach under different levels of dynamic workloads was tested using a 20 seconds, 0.05Hz sinusoid workload with a varying amplitude. Average system workload in these 20s was varied between 10% and 300% of total system capacity. The resulting query response times were normalized by dividing them with the respective ones of QA-NT. This is because most of the time, QA-NT found the best solution (It was not possible to find the exact optimal solution due to the complexity of the problem). The load balancing algorithms performed very poorly, therefore, in Figure 5a we show only the normalized response time of the Greedy algorithm.

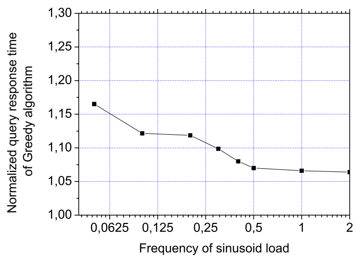
For small workloads (less than 75% of system capacity) Greedy performed roughly 5% better than QA-NT. The reason is that the (market) equilibrium amounts of Q1 and Q2 queries that each node should evaluate are very small *real* numbers, whereas our algorithm computes in each time period (500ms) an integer-valued number of Q1 and Q2 queries. This rounding procedure effectively leads our algorithm to error. For workloads above 75%, the significance of rounding errors is reduced and the query response time of the Greedy algorithm was 15%-32% worse than that of QA-NT. Furthermore, in additional experiments that we run, we found that if the number of query classes is large or the workload is static, then our algorithm converges close to market equilibrium and has superior performance in all possible workloads.

The above results show that our algorithm is especially recommended for distributed systems at times of high workload, when a good query allocation mechanism is most critical. QA-NT provides a native, decentralized way for nodes to understand when the whole systems is overloaded, which is when query prices are high. Thus, it is easy to implement QA-NT in a distributed system which will properly track query prices but will only use them to calculate the nodes' query supply vectors if they are above a specific threshold.

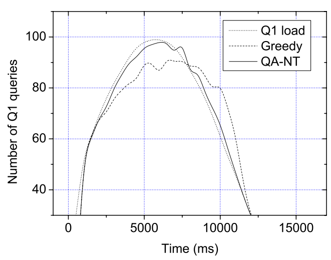
Figure 5b presents the behavior of our algorithm when the frequency of the sinusoid workload is varied between 0.05Hz and 2Hz. The average workload was 80% of total system capacity. In all cases, QA-NT proved that it can follow the dynamics of the load by performing better than



(a)



(b)



(c)

Figure 5: Behavior of QA-NT algorithm in dynamic environments.

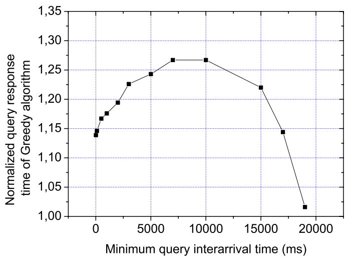


Figure 6: Results of simulated experimental study with workloads following a zipf distribution.

the Greedy algorithm. As expected, the improvement is reduced as frequency increases. Note that even a 0.05Hz sinusoid workload is difficult to be observed in real cases, as it represents a senario where total workload goes from 0% to 80% in just 10s.

Figure 5c gives details on how QA-NT and Greedy handle temporary loads close to the total capacity of the system. The graph shows the number of Q1 queries arriving per half second and the number of Q1 queries executed by QA-NT and by Greedy in the same time period. Since the Q2 queries sinusoid workload has a 90^0 degrees phase difference to that of Q1, in the presented time period (0-15,000ms) the algorithms had to handle the simultaneous allocation of both Q1 and Q2 queries, though the latter are fewer than the former. In the specific experiment, both algorithms perfectly allocated all Q2 queries (i.e., the least expensive ones) therefore, we show only the differences in the allocation of Q1 queries. Figure 5c shows that QA-NT manages to closely follow the presented load, whereas the Greedy algorithm overloaded the system and could not serve all Q1 queries. QA-NT allocated Q2 queries to the slower network nodes and thus left enough free resources to handle the Q1 queries. The Greedy algorithm did not make any such distinction between queries Q1 and Q2.

Heterogeneous workload Figure 6 summarizes our finding from running our simulator with the zipf workload of the second set of experiments. The figure presents the normalized query response time of the Greedy algorithm, when the minimum query inter-arrival time is varied from 10ms to 17,000ms. For small inter-arrival times (less than 5,000ms), QA-NT manages to improve the performance of the system by 13-24%, though the improvement falls as inter-arrival time is reduced (i.e., workload is increased). In moderate overload conditions (i.e., interarrival time arround 10,000ms) QA-NT improves system performanc by approximately 26%. Finally, as the interarrival time is further increased, the system recovers from the overload condition and thus, the gains of our QA-NT mechanism are reduced. For minimum inter-arrival times larger than 17,000ms, the system is not overloaded and QA-NT provides no improvement.

Note that a 10%-20% improvement is significant, as this equivalently means that the same workload can be handled by approximately 10%-20% less nodes. This is very important for very large federations of DBMSs.

5.2. Testing of a Real Implementation of QA-NT

Experiments setup

We implemented the Non-tâtonnement pricing mechanism in C++ and deployed it into 5 Windows PC nodes. These nodes were equipped with one or two processors with a speed of 1.3GHz-3.06GHz and 1Gbyte of main memory. Their network interconnection was based on a dedicated 100Mb full duplex Hub with the exception of one PC that was connected with a P2P 54Mb wireless connection.

The data of the experiments consisted of 20 tables that occupied 1Gb of tablespace and 80 select-project views over these tables. Each table/view had 2-4 copies. The workload consisted of select-join-project-group star-queries with an average execution time of approximately 1s in the fastest machine and 14s in the slowest one.

The implementations of both the Greedy and the QA-NT algorithms initially estimated the execution time of a query

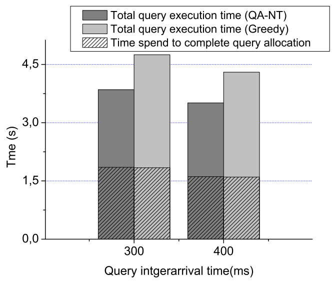


Figure 7: Results of experiments run over a small distributed network of RDBMSs.

using the `EXPLAIN PLAN` statement of the DBMS. Unfortunately, this estimation was usually incorrect as it did not take into account the contents of the DBMS buffers. Thus we ended up with the following algorithm: First the `EXPLAIN PLAN` was used to find the execution plan of the query and the relevant data statistics. Then, we used past execution information concerning queries with the same plan to estimate the execution time of the new query.

We run two experiments where 300 queries were evaluated using both Greedy and QA-NT. The query interarrival time had a uniform distribution with an average of 300ms and 400ms, respectively. We measured the time required by Greedy and QA-NT to assign a query to a node and the total query evaluation time (time to assign + execute query).

Results

Figure 7 shows the results of the two experiments run. In both cases, QA-NT performs better than Greedy. What is worth noticing is the relative long time it took for both algorithms to find the node that would actually execute incoming queries. This is because both algorithms waited for a reply from all nodes before deciding on the node assignment of queries. This caused large delays as the slowest of the PCs took up to 3 seconds to evaluate an `EXPLAIN PLAN` statement.

6. Conclusion

We have presented a microeconomics-based, decentralized query allocation mechanism, suitable for federations of autonomous database management systems. Our approach is unique in that it respects node autonomy and is compatible with existing distributed query optimization algorithms. We have run an extensive number of experiments using both a simulator and a real commercial system and compared our solution with existing load balancing approaches. In most cases, especially those of system overload, our algorithm has exhibited substantially superior performance compared to all alternatives.

In the future, we indent to extend our algorithm to consider the role of game theory and Baysian Nash equilibrium in query markets. In particular, we will introduce the constraint of *equitable allocation*, in which the utility (satisfaction) of all nodes is equalized. We will also examine the case of load balancing under uncertainty, the case of multiple discriminating queries properties (multi-objective), and the use of *insurance* from economic theory to ensure QoS.

References

[1] M. J. Carey, M. Livny, and H. Lu. Dynamic task allocation in a distributed database system. In *ICDCS*, pages 282–291, 1985.

[2] M. J. Carey and H. Lu. Load balancing in a locally distributed database system. In *SIGMOD Conference*, pages 108–119, 1986.

[3] S. Chaudhuri and G. Weikum. Rethinking database system architecture: Towards a self-tuning risc-style database system. In *VLDB 2000*, 2000.

[4] P. E. Drenick and E. J. Smith. Stochastic query optimization in distributed databases. *ACM Trans. Database Syst.*, 18(2):262–288, 1993.

[5] D. F. Ferguson. *The Application of Microeconomics to the Design of Resource Allocation and Control Algorithms*. Ph.d., Graduate School of Arts and Sciences, Columbia University, 1989.

[6] D. F. Ferguson, C. Nikolaou, J. Sairamesh, and Y. Yemini. *Economic Models for Allocating Resources in Computer Systems*. World Scientific, Hong Kong, 1996.

[7] H. Gravelle and R. Rees. *Microeconomics (3rd edition)*. Pearson Education, England, 2004.

[8] L. Liu, A. Reuter, K.-Y. Whang, and J. Zhang, editors. *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*. IEEE Computer Society, 2006.

[9] A. Mas-Colell, M. D. Whinston, and J. R. Green. *Microeconomic Theory*. Oxford University Press, New York, 1995.

[10] M. Mitzenmacher. How useful is old information. *IEEE Transactions on Parallel and Distributed Systems*, 11(1), January 2000.

[11] A. 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.

[12] K. Nakatsuka, H. Yamaki, and T. Ishida. Market-based network resource allocation with non-tatonnement process. In *Design and Applications of Intelligent Agents: Third Pacific Rim Int. Workshop on Multi-Agents (proceedings), PRIMA*, Melbourne, Australia, 2000. Springer-Verlag Heidelberg.

[13] F. Pentaris and Y. E. Ioannidis. Distributed query optimization by query trading. In *EDBT 2004*, 2004.

[14] F. Pentaris and Y. E. Ioannidis. Query optimization in distributed networks of autonomous database systems. *ACM Trans. on Database Systems*, 31(2):537 – 583, June 2006.

[15] M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeller, A. Sah, J. Sidell, C. Staelin, and A. Yu. Mariposa: A wide-area distributed database system. *VLDB Journal*, 5(1):48–63, 1996.