Performance of Adaptive Query Processing in the Mariposa Distributed Database Management System

Jeff Sidell
University of California, Berkeley
jsidell@cs.berkeley.edu

Abstract

The Mariposa distributed database management system is based on an economic paradigm in which processing sites buy and sell resources such as CPU time, I/O capacity and network bandwidth [STO96]. In a Mariposa economy, a site assigns a price to perform a service and may adjust its prices as it sees fit. In designing Mariposa, it was our hope that market forces could be used to achieve good performance while maintaining the independence of each processing site. In this paper, we address the problem of distributed query processing in Mariposa and compare its performance to a traditional cost-based distributed query optimizer. We demonstrate the ability of a Mariposa system to adapt to a dynamic workload by using simple economic concepts such as the effect of supply and demand on prices. We investigate the benefits of adaptive distributed query processing in Mariposa and its interaction with multi-user workloads, network latencies and query size. We present performance results which show that in multi-user situations, when response time is used as a metric, the Mariposa system outperforms a static optimizer by causing work to be distributed more evenly among the available sites and that the overhead introduced by Mariposa's bidding protocol is insignificant when used with large, expensive queries and is outweighed by the benefits of load balancing, even for relatively small queries. Our experiments demonstrate that the point at which our approach outperforms a static optimizer is affected by network latency and query size. Our performance comparisons are based on the TPC-D benchmark.

1. Introduction

The Mariposa distributed database management system is based on an economic paradigm in which processing sites buy and sell resources such as CPU time, I/O capacity and network bandwidth [STO96]. In a Mariposa economy, a site assigns a price to perform a service and may adjust its prices as it sees fit. In designing Mariposa, it was our hope that market forces could be used to achieve good performance while maintaining the independence of each processing site.

In this paper, we address the problem of distributed query processing in Mariposa and compare its performance to a traditional cost-based distributed query optimizer. A traditional cost-based distributed optimizer creates a query plan which specifies not only the join order and access methods but also the processing sites at which each operation will take place. Such an optimizer will produce the same plan for a query regardless of current conditions such as available buffer space or network traffic. For this reason, we call these types of optimizers *static*. A canonical example of a static optimizer is R* [WDH81]. Static optimizers are unable to react to fluctuating availability of resources by adjusting the relative values of those resources in their cost functions. A simple example will illustrate this point: A company has two database servers on its network: one in the accounting department which contains employee data and the other in the sales department with customer data. During payroll processing, the accounting department server experiences a spike in its disk, CPU and memory usage while the sales department server is relatively idle. A static optimizer will continue to produce plans that run exclusively on the accounting department server, even though some work, such as sorting, joins, and aggregation could be done by the other machine, decreasing average response time. While it may be relatively straightforward to rewrite a static optimizer to fix this particular problem, if the two servers are running different database management systems on different hardware platforms with different operating systems, this is not an option. Furthermore, if the accounting department moves

across the country or upgrades its server hardware, the optimizer must be reconfigured. A system which adapts automatically to changing situations and is implemented in middleware is a better approach.

In this paper, we demonstrate the ability of a Mariposa system to adapt to a dynamic workload by using simple economic concepts such as the effect of supply and demand on prices. We investigate the benefits of adaptive distributed query processing in Mariposa and its interaction with multi-user workloads, network latencies and query size. We present performance results which show that in multi-user situations, when response time is used as a metric, the Mariposa system outperforms a static optimizer by causing work to be distributed more evenly among the available sites and that the overhead introduced by Mariposa's bidding protocol is insignificant when used with large, expensive queries and is outweighed by the benefits of load balancing, even for relatively small queries. Our experiments demonstrate that the point at which our approach outperforms a static optimizer is affected by network latency and query size. Our performance comparisons are based on the TPC-D benchmark [TPC].

The next section describes previous work in the areas of distributed query optimization and parallel and distributed query processing. Section 3 describes the Mariposa architecture, paying particular attention to the query broker and bidder modules. A complete description of the Mariposa architecture can be found in [STO96]. We describe the experimental setup in Section 4, and in Section 5 we present the results. We propose directions for future work in Section 6.

2. Previous Work

Query optimization and query processing have been the subject of a great deal of research, starting with traditional single-site cost-based optimization [SEL79]. When distributed database management systems were first introduced [WDH81] [BER81] [STO86] the single-site cost-based optimizers were changed to take into account network costs. Conventional distributed (static) query optimizers consist of a cost function, which estimates resource usage such as network cost, CPU time and disk I/O, based on input parameters such as relation size and selectivity of join operations. Static optimizers make assumptions about such things as network and hardware homogeneity and availability of storage and CPU resources. A static optimizer will attempt to minimize the cost function using an algorithm developed with the set of assumptions in mind. The cost function used by a static optimizer is usually generated once when the system is installed, and reflects the hardware and network configuration of the system.

The parameterized query optimizer presented in [IOA92] is an attempt to address the problem of adapting to dynamic resource allocation. In a parameterized query optimizer, many possible plans are generated and one is selected at run-time, depending on the values of parameters such as buffer size. However, this approach is limited in its scalability: The number of query plans that would need to be generated to take into account all possible combinations of buffer space, CPU usage, disk traffic, network traffic, data layout, hardware configuration and user-defined functions is potentially enormous.

An approach closely related to that taken by Mariposa is presented in [DAV95]. The authors address the problem of multiple query management in a single-site database by utilizing a resource broker, which sells resources to competing operators. This work has many similarities to Mariposa, including a microeconomic model where the broker's goal is to maximize its profit. However, their work has not been expanded to distributed databases.

Transaction processing monitors [GRA93] also include distributed load balancing as part of a long list of services. When a request arrives from a client, the TP monitor decides whether to execute the request immediately, queue it to run as soon as a server process becomes available, queue it to run later, or send it to a remote node for execution. Because TP monitors were designed for systems that process many small transactions, the decision must be made quickly. Most TP monitors utilize a few heuristics, which are installed at run time and cannot be modified. In contrast, Mariposa's brokering and bidding process is more expensive but also more flexible. For example, each Mariposa bidder can formulate its bid in any way it chooses.

Much research in parallel query processing has focused on speeding up single queries, primarily by exploiting intra-operator parallelism [MD95]. An operation which can be performed in parallel by several processors at once, such as sorting [DNS91] or hash joins [ZG90] is divided among all available processors. Intra-operator parallelism is sensitive to data skew; since each processor is performing essentially the same task over different data, it is important that the division of data among the processors be as close to even as possible. Overcoming data skew has been studied extensively and the various approaches are well-documented in the literature [WDJ91] [WDY91] [DNS92] [HLY93]. [RM95] proposes dynamic load balancing schemes for parallel join processing and compares them to static algorithms in a simulated multi-user environment. The dynamic load balancing techniques adjust the degree of join parallelism based on several factors, such as disk I/O, CPU usage and memory usage. The dynamic techniques presented in the paper outperformed static algorithms, which assign a fixed number of processors to parallel joins. The Mariposa model is general enough to represent intra-operator parallelism: work (in this case, a single operator such as a sort) could be divided among several machines. This is a subject for future research. The work presented in this paper deals with a much more coarse-grained parallelism, namely inter-operator and inter-query parallelism, but not intra-operator parallelism.

Whereas a static optimizer will produce a query plan, complete with processing sites, which minimizes its cost function, Mariposa first produces a query plan, then divides the work to be done among the available processors based on their current resource availability. This is a generalization of the approach taken by some parallel database management systems [HON92] [HAS95] which perform *join order query rewrite* (JOQR) first, then divide the resulting plan among the available processors based on their current load, assigning the most work to the least-burdened processor. During JOQR, the query is optimized, producing a query plan, which is a tree of operation nodes. Each node represents a piece of work such as scanning a relation, sorting, or performing a join. The XPRS parallel database management system [HON92] used a single-site optimizer during this phase and therefore ignored communication cost. [HM95] presents algorithms that incorporate communication costs.

Dividing a plan into parts and scheduling the parts in an optimal way is itself an NP-complete problem. [CHM95] presents two approximation algorithms for dividing query plans into subplans for scheduling on a parallel machine. The original query plans must consist only of non-blocking operators such as sorts and hash table builds. We plan to implement and test these approaches to dividing query plans into subplans as future work. The algorithms will have to be expanded to deal with blocking operators and network latency.

3. Background

Mariposa was designed as middleware; it is intended to sit between a single-site DBMS and a front-end application. The following example is illustrated in Figure 1, which shows the Mariposa modules and the communication among sites during query processing. For a detailed description of the Mariposa architecture, refer to [STO96] or [MAR96]. A front-end application submits a query such as

```
SELECT O_ORDERPRIORITY, count(O_ORDERKEY) AS ORDER_COUNT
FROM ORDERS, LINEITEM
WHERE           O_ORDERDATE >= 01/01/1997
   AND    O_ORDERDATE < 01/16/1997
   AND    L_ORDERKEY = O_ORDERKEY
   AND    L_COMMITDATE < L_RECEIPTDATE
GROUP BY O_ORDERPRIORITY
ORDER BY O_ORDERPRIORITY;
```

and a *bid curve* at the *home site*. The bid curve has *cost* on the y axis and *time* on the x axis and specifies how much the user is willing to spend to have his or her query processed within a given amount of time. The query passes through a parser, optimizer and fragmenter and is turned into a query plan. The parser, optimizer and fragmenter (and, later, the query broker) use information from a Mariposa *name server.* Name servers provide system metadata including type information, data fragmentation and placement. Name service in Mariposa is described in [SID96]. In Mariposa, the optimizer is a single-site cost-based optimizer which minimizes single-site resources used to execute the query. The query plan produced by the optimizer is passed into the *fragmenter*. The fragmenter alters the single-site plan to reflect the underlying data fragmentation. The experiments in this paper were not performed over fragmented data so we will leave out a detailed explanation of the fragmenter. The interested reader can refer to [STO96].

The query plan is passed into the *query broker*, whose job it is to assign a processing site to each node in the plan tree. First, the query broker breaks up the plan into *plan chunks*. A plan chunk can be as small as a single node or, as indicated in Figure 1, as large as the entire query plan. Plan chunks are the basic unit of work assigned to a processing site. Dividing a plan into many small chunks increases the potential for parallel and pipelined execution of the plan, while dividing it into a few large chunks decreases potential parallelism and pipelining. In the experiments in this paper, the broker sent out whole plans without breaking them up (although parts of plans were **subcontracted** by bidders to other sites - see Section 3.1). The effects on query processing of breaking up plans in different ways is an area for future study.

The query broker follows one of two protocols: long or short. The experiments in this paper all used the long protocol. For a description of the short protocol, refer to [STO96]. In the long protocol, illustrated in Figure 1, the query broker contacts *bidder* processes running at potential processing sites, passing along a plan chunk and soliciting a bid. Each bidder responds with a bid which contains the cost and time the bidder site will require to perform the work specified. In Figure 1, the broker has asked two potential processing sites to bid on the plan chunk indicated, and gotten back two bids: **($5, 5 minutes)** and **($10, 30 seconds)**. The bidder's response is determined completely by a script written in an enhanced Tcl [OUS90]. The broker selects the bid for each plan chunk which corresponds to the point which is farthest below the bid curve. Once the broker has determined the processing sites, the distributed plan is passed to a *coordinator* module, which contacts the processing sites to begin execution. In

Figure 1, the work has been awarded to the Gray Site. After a processing site has completed its work, it is paid the amount agreed to during the bidding phase.

Adding network costs into plans on the broker site and then choosing the sites that result in the best plan (i.e. the one farthest under the bid curve) has exponential growth; the number of choices is equal to (number of bidder sites) $^{(\text{number of plan chunks})}$. As long as plan chunks are relatively large and there are relatively few bidding sites, this is not a problem. However, if plan nodes for a large plan were bid out individually even to a relatively small number of sites, the solution space becomes too large. For example, the TPC-D queries can produce plans with up to 60 or so nodes. If each node were bid out individually to ten sites, there would be 10^{60} possible solutions. To address this issue, we intend to pursue the work mentioned in Section 2 on optimal division of plan trees [CHM95].

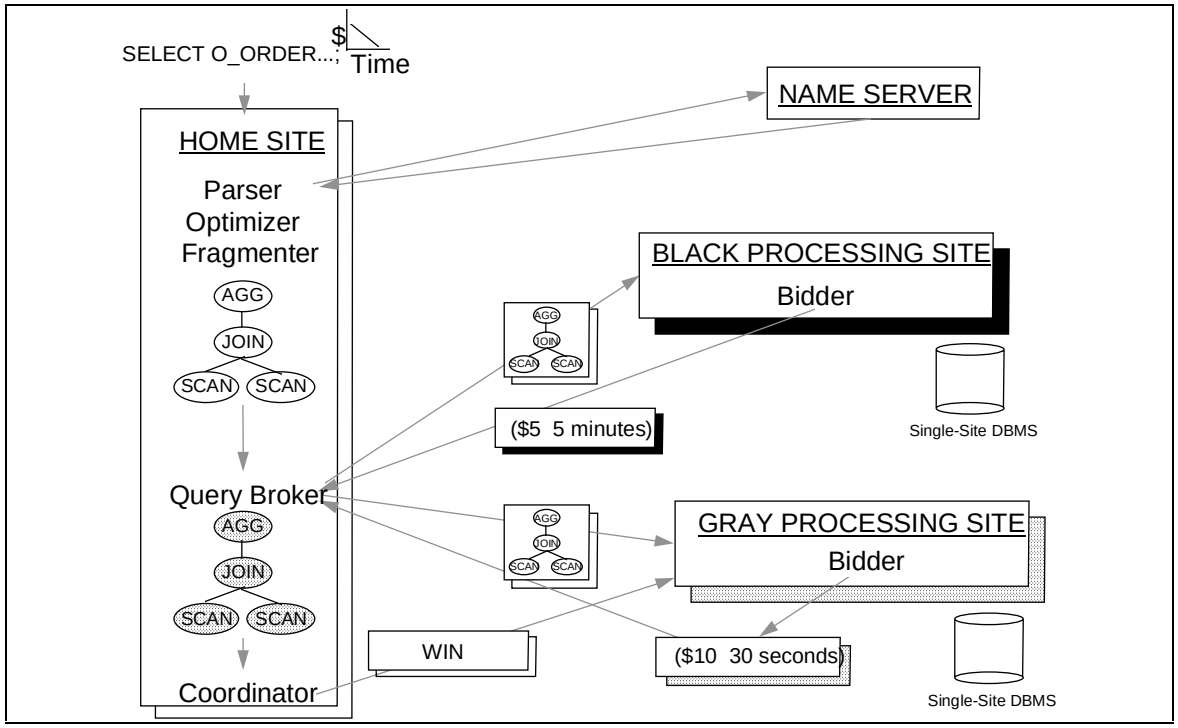

Figure 1: Mariposa Architecture

3.1 The Bidder and Subcontracting

One important Tcl extension we have made for the bidder is the **Subcontract** command. It is possible that a bidder may be able to process part but not all of a subquery it is sent by the broker. For example, if the plan in Figure 1 were received by a bidder site which only had the LINEITEM relation but not the ORDERS relation, it could process all of the plan except the scan over ORDERS. In this case, the bidder can contact its own broker to solicit sub-bids from other bidder sites for the part of the plan it cannot process itself. It adds the best sub-bid into the bid it returns to the original broker site. See Figure 2, in which the Black site subcontracts out the scan of ORDERS to the Gray site. Subcontracting has the added benefits of distributing the brokering costs and allowing the broker at the home site to send out whole plans to be bid on without concern for data layout, user-defined functions or other site-specific

capabilities. Another benefit of the subcontracting mechanism is that it allows network costs to be factored into the bid sent back from the original bidder site (such as the Black site in Figure 2). This eliminates the problem mentioned in Section 3 of the broker adding in network costs. When a bidder subcontracts, it knows that the site that will process the subcontracted work is different than itself and so can factor in the network cost.

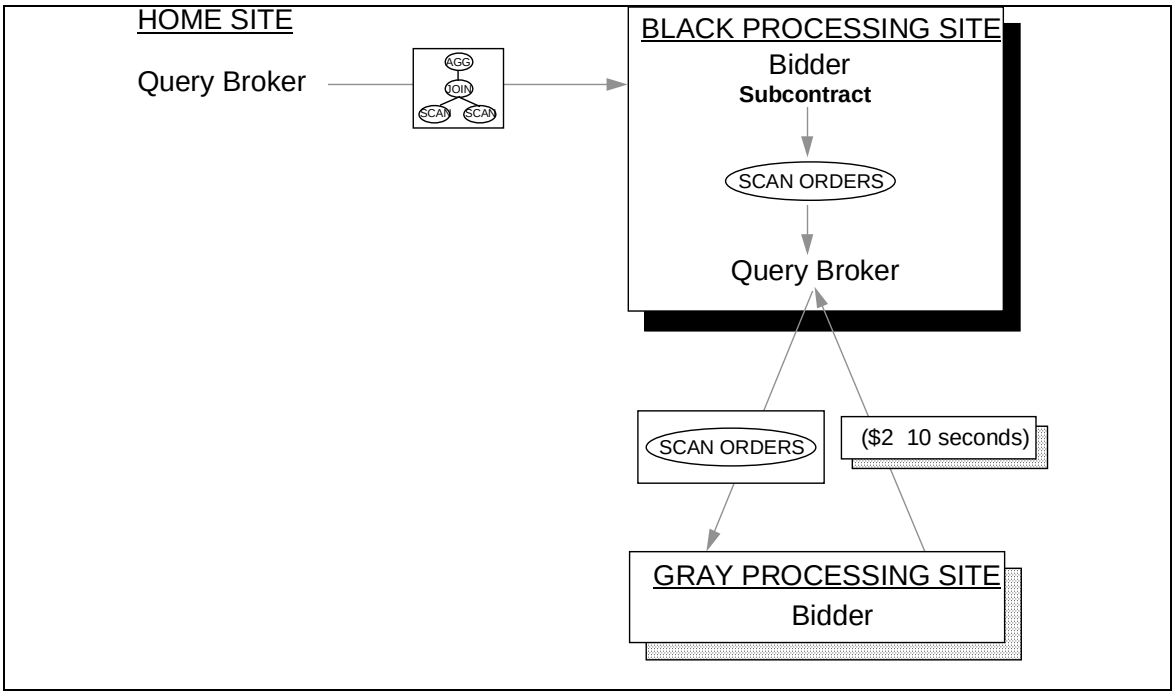

Figure 2: Subcontracting

4. Experimental Setup

The performance studies are based on the Transaction Processing Council's benchmark D [TPC]. This benchmark is designed to model decision-support queries. There are seventeen queries over nine database tables (including one optional table). Several of the queries are multi-way joins and all of them include aggregates. The database size was 10MB (corresponding to a TPC scale factor of 0.01) in all of the experiments, except in Section 5.3 where we tested smaller table sizes. The experiments were set up as follows:

- We ran the benchmarks on three DECStation 3000/300's running at 150 MHz. Each workstation has one Alpha/21064 processor and at least 64MB of RAM. Each machine had between 250MB and 1GB of available disk space.
- Network packet size = 8192 bytes, of which 1024 bytes was header and 7168 bytes was data. Disk page size = 8192 bytes.
- The network protocol used in Mariposa is ONC RPC over TCP/IP. Average overhead for each remote procedure call was 49 milliseconds. This overhead includes the extra instructions at each site involved in making the remote procedure call, as well as the actual network transmission time.
- All the experiments were run over an FDDI network with a bandwidth of 100 Mb/second.
- We varied the number of concurrent users, starting with one and increasing up to ten. Each user ran each of the seventeen TPC-D queries once, in random order.

- We created what we considered to be a reasonable data layout as follows: The size of each table was multiplied by the number of times the table was scanned during the seventeen TPC-D queries to arrive at a weight as shown in Table 1. The tables were assigned to machines to distribute the weight most evenly. Even so, one workstation (Chorizo) ended up with the largest and most heavily-used table.

TABLE	SIZE (bytes)	Number of Queries	WEIGHT (scaled)	Server
LINEITEM	11,640,832	14	1629.72	Chorizo
PARTSUPP	1,744,896	4	69.80	Parkcity
NATION	8,192	7	0.57	Parkcity
CUSTOMER	376,832	5	18.84	Parkcity
REGION	8,192	2	0.16	Parkcity
PART	442,368	6	26.54	Parkcity
SUPPLIER	24,576	9	2.21	Parkcity
TIME	262,144	5	13.11	Parkcity
ORDERS	2,736,128	9	246.25	Utopia

Table 1: Data Layout for Experiment #1

5. Experimental Results

5.1 Mariposa vs. a Static Optimizer

In the first experiment, we compared Mariposa to a static distributed optimizer. To create the static optimizer, we enhanced a single-site optimizer's cost function to include network costs. The network costs included the additional CPU time for connection setup and teardown and the per-tuple CPU cost, as well as the per-packet network cost of transmitting the data. The static optimizer was allowed perfect information about relation sizes, selectivity of WHERE clauses and cardinality of join results. It exhaustively considered all distributed plans and chose the plan which minimized its cost function. Optimization time was not included in query processing time. The query broker was turned off for this run, since the processing sites were determined by the static optimizer.

After we ran the benchmark queries with the static optimizer, we turned the query broker back on. The optimizer produced only single-site plans, ignoring network overhead. The query plans produced by the single-site optimizer were sent to bidder sites in their entirety and each site was allowed to subcontract those nodes of the plan it could not execute, namely scans over tables it did not own. When a site subcontracted a table scan to another site, it added the network cost into the total bid. Bidding out entire plans (rather than, say, individual nodes) minimized the bidding overhead, but meant that the granularity of work assigned to each site was very coarse. In contrast to the static distributed optimization time, the elapsed time for the brokering and bidding process was included in the overall query processing time.

The bidder for this experiment used LA_{60}, the 60-second system load average (average number of jobs in the run queue) as a crude estimate of resource consumption. The bidder at each site recursively descended the plan tree, assigning a cost and a time estimate to each node and adding them to arrive at a cost-based bid. The cost was then

multiplied by $(1 + LA_{60})$. The bid curve for these experiments was horizontal, which meant that work was awarded to the site which bid the lowest cost. This strategy had the effect of causing an entire query plan, minus scans over non-local base relations, to be executed on the site with the lowest 60-second load average.

Figure 3 shows the average response time per query for queries run with the static optimizer and with Mariposa's query broker. The static optimizer performed slightly better on a single query than the broker. This is to be expected, since the static optimizer could consider network cost in assigning processing sites to the nodes in the plans, whereas Mariposa's single-site optimizer does not. Processing an entire query (except non-local table scans) on the site with the lowest 60-second load average is likely to generate more network traffic, and therefore take longer, than the plan produced by the static optimizer. Since queries are processed serially, there is no inter-query parallelism.

When the number of users increased, the query broker outperformed the static optimizer. The slope of the two lines indicates that the average response time per query for the static optimizer will continue to degrade more quickly than that for the query broker. By causing sites which are less busy to process work that the static optimizer assigned to a busier site, and thereby distributing the workload more evenly, Mariposa decreased the average response time.

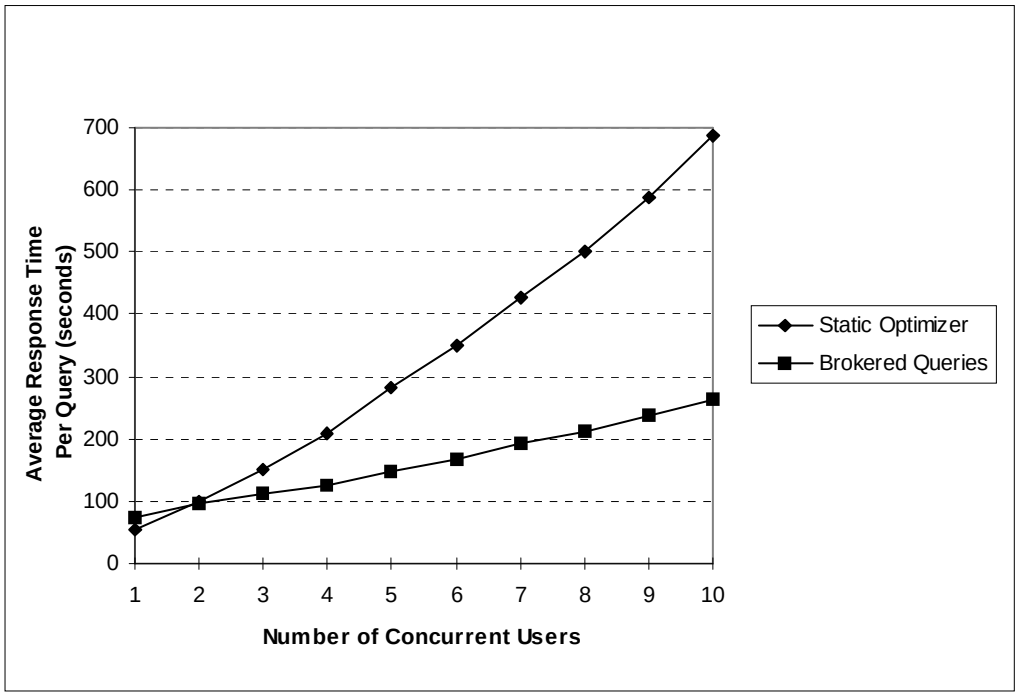

Figure 3: Average Response Times for Mariposa Brokered Queries vs. a Static Optimizer

5.1.1 Workload Distribution

In order to see how evenly Mariposa distributed work among the processing sites, we measured the amount of work performed by each site to run queries produced by the static optimizer and compared it to the work to process brokered queries. Figure 4 shows the workload distribution among the three servers for the static optimizer, and Figure 5 shows the same thing for Mariposa brokered queries. The X axis shows the number of concurrent users, as

in Figure 3. The Y axis shows the percentage of the total workload performed by each server. This percentage was calculated as follows, using parkcity as an example:

$$\%WORKLOAD_{parkcity} = (\%CPU_{parkcity} + \%DISK_{parkcity} + \%NET_{parkcity}) / 3$$

where:

$$\%CPU_{parkcity} = (CPU_{parkcity}/(CPU_{parkcity} + CPU_{utopia} + CPU_{chorizo}))$$

$$\%DISK_{parkcity} = (DISK_{parkcity}/(DISK_{parkcity} + DISK_{utopia} + DISK_{chorizo}))$$

$$\%NET_{parkcity} = (NET_{parkcity}/(NET_{parkcity} + NET_{utopia} + NET_{chorizo}))$$

and:

$CPU_{machine}$ = total CPU time used on *machine* over all queries

$DISK_{machine}$ = total number of disk blocks read and written on *machine* over all queries

$NET_{machine}$ = total number of network packets sent and received by *machine* over all queries

Figure 4 shows that the workload was distributed among the machines consistently by the static optimizer: each machine was given the same percentage of the total workload to perform, regardless of the total amount of work. This is to be expected, since the static optimizer produces the same plan for a query, regardless of current conditions, and each user ran the same queries, just in a different order. Chorizo has a much higher percentage of the total workload, since it contains the largest, most heavily-used table. Chorizo therefore became a bottleneck and slowed down query processing.

Figure 5 shows that the workload was distributed more evenly among the three machines by the Mariposa query broker. Chorizo still performed more work than the other two machines, but the difference among the machines is far less than for the static optimizer. For a single user, the workload distribution is similar to that for the static optimizer. For two users, the distribution is closer to even, and for three and more users, the distribution has become still more even, with the difference between the work performed by Chorizo and that performed by the least-burdened site, Parkcity, remaining at about 15 percent.

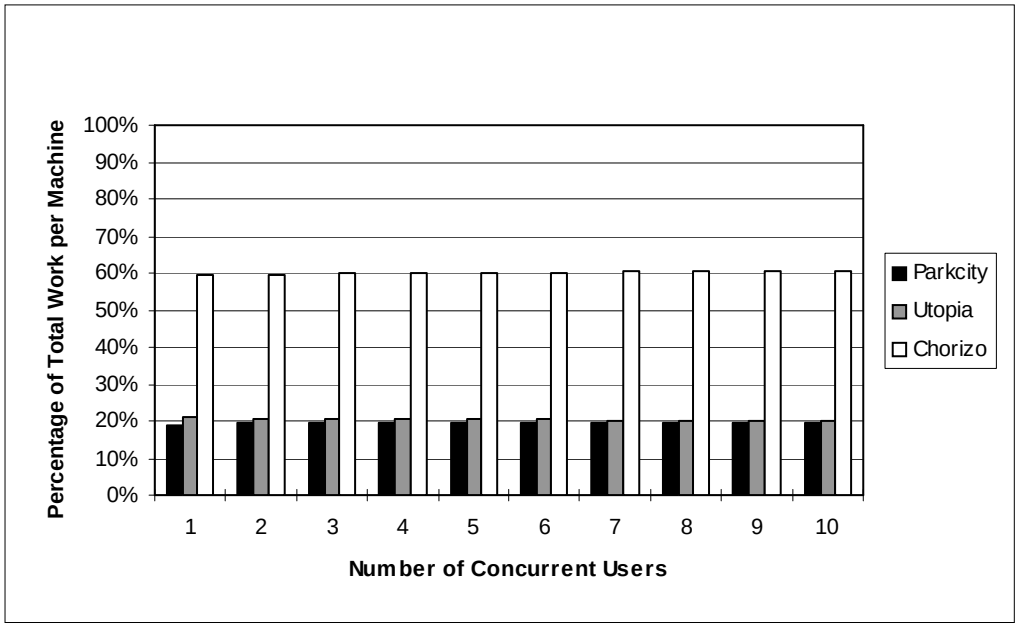

Figure 4: Workload Distribution for Static Optimizer

It is true that load balancing could also be achieved by other means, including fragmenting the large tables and distributing them among the three sites. This is a perfectly reasonable approach and is likely to obtain good results for some queries. However, fragmenting the tables and distributing the fragments does not guarantee load balancing: one fragment could be much more heavily-used than the others. The Mariposa approach, being adaptive, has a greater likelihood of achieving load-balancing. Furthermore, the approach taken by Mariposa is much more general: the tables involved in a query can be under separate administrative domains. In such a situation, fragmentation is not an option. We plan to address query processing with fragmented tables as future work.

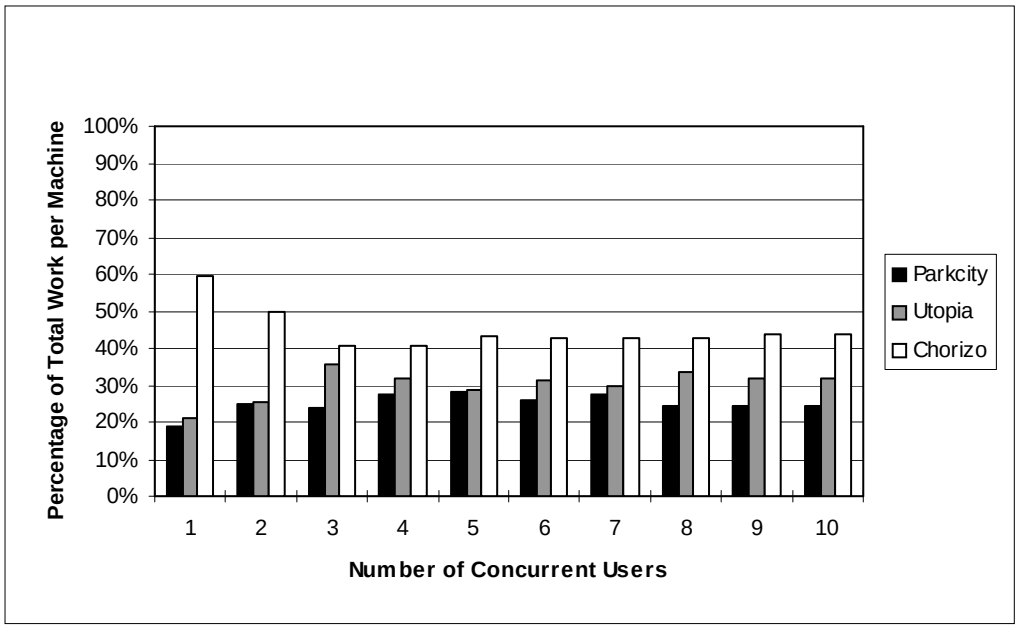

Figure 5: Workload Distributed for Mariposa Brokered Queries

5.1.2 Query Brokering Overhead

To measure the overhead of communication among the three sites due to the query brokering process, we measured the elapsed time from the beginning of the bidding process until it was completed and the processing sites started work. Table 2 shows the communication overhead for brokering vs. the average query response time. Clearly bidding adds only negligible overhead - less than one percent of the overall response time.

Number of Concurrent Users	Average Response Time Per Query (secs)	Average Brokering Time per Query (secs)	Percentage of Total Response Time for Brokering
1	75.08	0.83	1.10%
2	94.81	1.52	1.61%
3	111.02	1.06	0.95%
4	125.22	1.31	1.05%
5	149.14	1.27	0.85%
6	167.95	1.33	0.79%
7	192.58	1.55	0.80%
8	211.49	1.55	0.73%
9	238.00	1.21	0.51%
10	263.97	1.73	0.65%

Table 2: Average Response Time and Bidding Time Per Query

5.2 Network Latency

Because of the overhead imposed by the brokering process, and its use of a single-site optimizer to generate plans, Mariposa's performance is sensitive to network delay. To test Mariposa's performance on slower networks, where network latency represents a significant part of query processing time, we repeated the experiment described in Section 5.1 with increased network latency, introducing artificial delay for network messages and for data transfers. Each time a remote procedure call was made or data was transmitted to a remote machine, the sending machine delayed for a fixed period of time. We used an additional network latency of 110 milliseconds, which was the average increase in the time for a remote procedure call observed when running the TPC-D queries among UC-Berkeley, UC-Santa Barbara and UC-San Diego. We also observe that this average latency was close to the median latency among the three sites of 86 milliseconds. 110 milliseconds is greater than the latency that could be expected from bandwidth limitations; the average observed bandwidth among the three campuses was 356 Kbps. Mariposa data streams are sent in 8K packets, resulting in an expected delay due to bandwidth limitations of 22ms. It was our original intention to use the results from running the test queries among the three remote sites. However, because of the wide variability of network latency, we could not obtain reproducible results. For this experiment, we changed the static optimizer's cost function to account for the increase in network latency. Similarly, we increased the network cost added in to a Mariposa bidder's price during subcontracting.

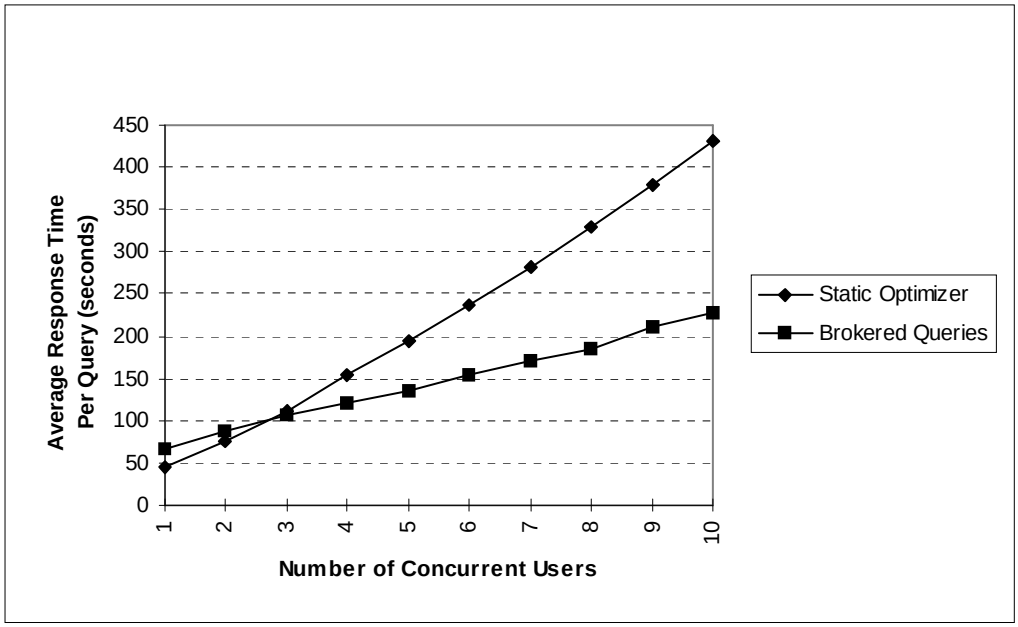

Figure 6: Average Response Times for Mariposa Brokered Queries vs. a Static Optimizer with 110ms Network Latency

The average response times for queries run with the static optimizer and with the Mariposa query broker are shown in Figure 6. Because of the additional network latency, and the increase in the cost of performing remote scans, the query broker was more likely to assign a query to be processed at the site which contained most of the data used in the query. This site (the one which contained the LINEITEM table) would continue to perform more work until its load average had increased to a greater point than when the benchmark was run over a faster network. Furthermore, there is a greater penalty if the single-site optimizer chooses a join order which would necessitate excessive data movement. The static optimizer could avoid such plans. In Figure 6, the query broker does not begin to outperform the static optimizer until there are three users. Over a faster network, as described in Section 5.1, the query broker outperformed the static optimizer when there were only two users. As the network latency among the processing sites increases, the point at which Mariposa will outperform a static optimizer also increases. [1]

5.3 Variations in Query Size

The queries described in the previous sections represent substantial amounts of work. On smaller queries, Mariposa's brokering overhead will represent a larger percentage of the query processing time. Furthermore, the advantage to be gained by more careful selection of processing sites is smaller when the amount of work represented by a query is small. To test the value of Mariposa's brokering strategy on smaller queries, we repeated the experiment described in Section 5.1 with TPC-D scale factors of 0.001 and 0.0001. The table sizes for these scale factors are shown below in Table 3. All sizes are in bytes. The minimum table size is 8192, corresponding to one disk page. All other factors remain as described in Section 5.1.

[1] Observant readers will have noticed that the latencies in Figure 6 are smaller than those in Figure 3, even though additional network overhead was introduced. For the network delay experiments, one of the workstations used in the original experiments was replaced with one that had twice the available buffer space.

TABLE	Scale Factor 0.001	Scale Factor 0.0001
LINEITEM	1,163,264	114,688
PARTSUPP	180,224	24,576
NATION	8,192	8,192
CUSTOMER	40,960	8,192
REGION	8,192	8,192
PART	49,152	8,192
SUPPLIER	8,192	8,192
TIME	262,144	262,144
ORDERS	278,528	32,768

Table 3: Database Table Sizes for Scale Factors 0.001 and 0.0001

The average response time for queries run with the static optimizer and with the Mariposa query broker for TPC-D scale factors 0.001 and 0.0001 are shown in Figures 7 and 8, respectively. Because the amount of work represented by each query is smaller than for the experiments described so far, the performance degradation for each additional user is less. Still, Mariposa outperforms a static optimizer with relatively few users in each case, indicating that the bidding overhead, shown in Tables 4 and 5, is more than compensated for by load balancing. It should be noted that, although the amount of work represented by each query is small, it is still significant; A multi-way join, even over small tables, still consumes considerable CPU time, disk I/O and other resources. It would be interesting to test Mariposa on other benchmarks with smaller queries, such as the Wisconsin Benchmark [BIT83]. We plan to compare Mariposa to a static optimizer on the Wisconsin Benchmark as future work.

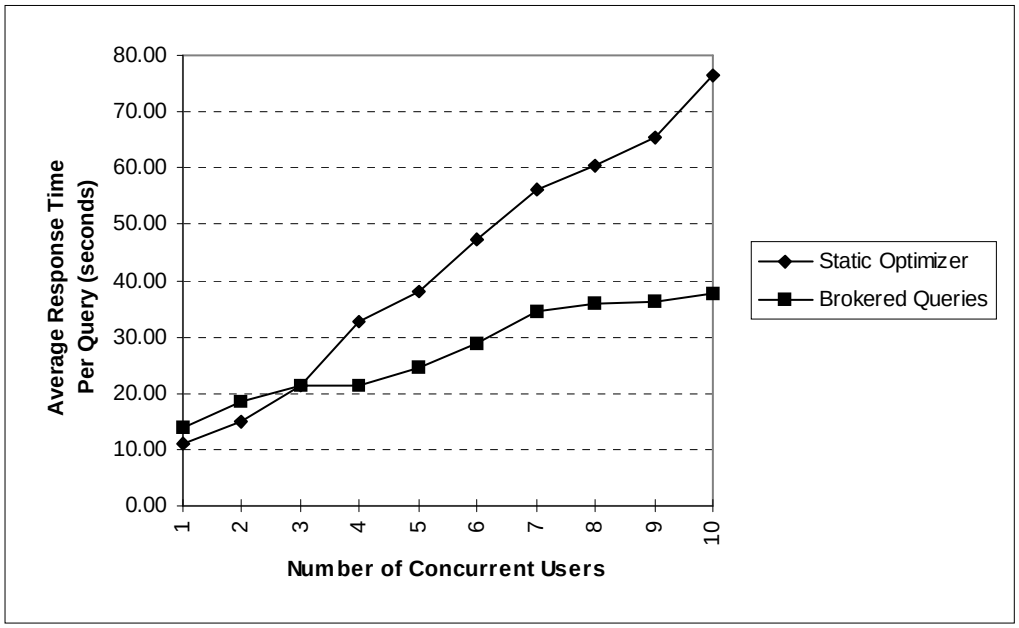

Figure 7: Average Response Times for Mariposa Brokered Queries vs. a Static Optimizer for TPC-D Scale Factor 0.001

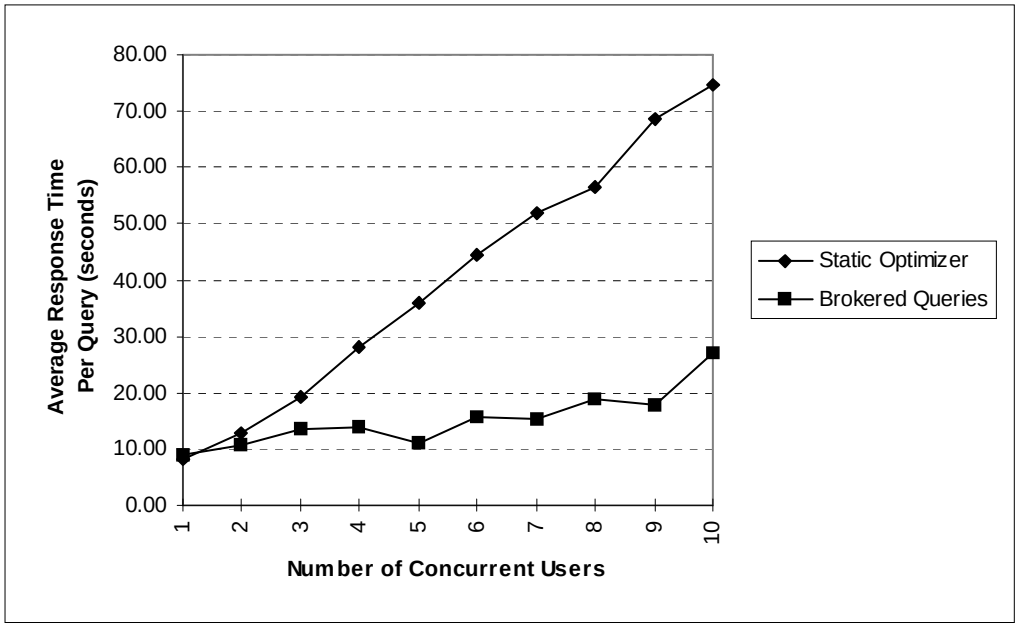

Figure 8: Average Response Times for Mariposa Brokered Queries vs. a Static Optimizer for TPC-D Scale Factor 0.0001

Number of Concurrent Users	Average Response Time Per Query (secs)	Average Brokering Time per Query (secs)	Percentage of Total Response Time for Brokering
1	13.84	0.87	6%
2	18.54	1.26	7%
3	21.39	1.43	7%
4	21.41	1.50	7%
5	24.42	1.34	5%
6	28.79	1.49	5%
7	34.57	1.38	4%
8	35.77	2.35	7%
9	36.23	2.16	6%
10	37.83	2.21	6%

Table 4: Average Response Time and Bidding Time Per Query for TPC-D Scale Factor = 0.001

Number of Concurrent Users	Average Response Time per Query (secs)	Average Brokering Time per Query (secs)	Percentage of Total Response Time for Brokering
1	9.03	1.46	16%
2	10.63	1.38	13%
3	13.38	1.52	11%

4	13.83	1.72	12%
5	11.14	1.20	11%
6	15.49	1.43	9%
7	15.36	1.69	11%
8	18.67	1.88	10%
9	17.66	2.76	16%
10	27.09	2.05	8%

Table 5: Average Response Time and Bidding Time Per Query for TPC-D Scale Factor = 0.0001

5.4 Speedup

In addition to balancing the load among machines that store database tables, and which therefore would be involved in query processing in any case, Mariposa can utilize any machine that is part of its "economy" - that is, any machine that is running Mariposa and has registered its existence with the rest of the system. Thus, in the example in Section 1, the accounting department server should be able to offload work to the sales department server. To test the speedup that Mariposa can obtain by using otherwise idle machines, we put all the tables on a single machine and ran the benchmark queries, increasing the number of users from one to ten. The TPC-D scale factor for this experiment was 0.01, as in Sections 5.1 and 5.2. We then added a second machine and re-ran the benchmark. We repeated the experiment with three machines. The data layout was not changed during the course of this experiment: the extra machines were used only to perform sorting, joins, aggregation and other such operations. As in Section 5.1, plans were bid out in their entirety and table scans were subcontracted if they could not be performed locally. The bidders formulated their bids in the same way as described in Section 5.1. We did not perform this experiment with the static optimizer.

Figure 9 shows the average response times when one, two and three machines were made available. The Mariposa query broker was able to use the additional machines to decrease the average response time. It was not necessary to compare these results with a static optimizer: in the case where all the tables are stored at one site, the lowest-cost plan is the one which performs all the work at that site.

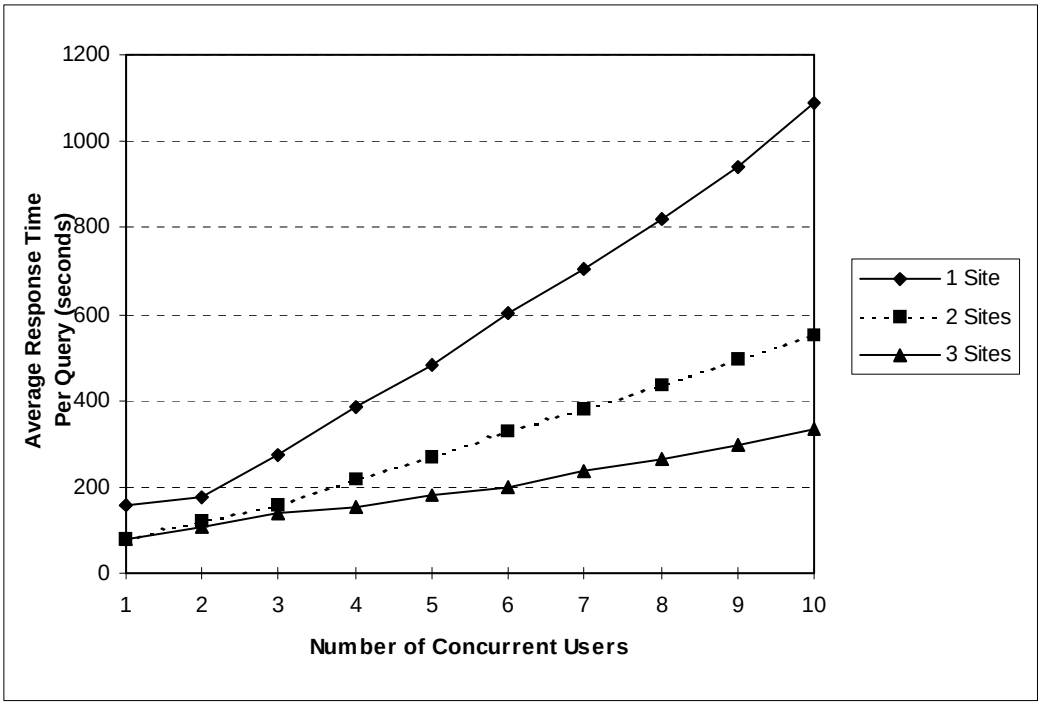

Figure 9: Average Response Time for Mariposa Brokered Queries with 1, 2 and 3 Available Processing Sites

The average speedup for two and three machines over one machine is shown in Table 6. The overhead imposed by the additional communication during brokering and data transfer is about ten percent, calculated as (1 - average speedup)/number of machines. The effect of pipelined parallelism is apparent when we observe that with one user, when queries are submitted serially so there is no inter-query parallelism, there is speedup with two machines. While the machine which stores all the base tables is scanning a table, the second machine can be performing work in parallel.

No. of Sites	Average Speedup
2	1.81
3	2.63

Table 6: Speedup for 2 and 3 Sites with Mariposa Brokered Queries

6. Conclusions and Future Work

The results presented in this paper show that Mariposa is capable of using its economic paradigm to perform load balancing and thereby outperform a static distributed query optimizer. Furthermore, we have shown that the overhead required for Mariposa's bidding protocol is insignificant when processing large queries and is overshadowed by the benefits of load balancing even for relatively small queries. This is a specific application of a very general framework. Our performance results show that Mariposa provides a solid basis from which we can investigate other applications of the economic paradigm. The bidder algorithm used for these performance studies is

quite primitive: load average is only a crude estimate of overall resource consumption. We intend to develop more sophisticated approaches to bidding which take into account individual resource availability. We also intend to expand on the work presented in [CHM95] to develop improved algorithms for dividing query plans up into units of work.

As mentioned previously, the Mariposa model is general enough to support intra-operator parallelism. However, the Mariposa model is capable of a more intelligent approach to dividing work among machines in a shared-nothing environment. Instead of dividing the work evenly, as in a parallel system where the processors are otherwise idle, Mariposa would assign a piece of work to a server in proportion to the site's ability to do the work, thus assigning the most work to the least-busy server. In situations where the amount of work cannot be accurately predicted in advance because of data skew, such as the probe phase of a distributed hash-join, Mariposa can use its subcontracting mechanism to distribute work more evenly among available processors.

We are in the process of porting Mariposa to Sun Solaris on the SPARC and Intel architectures, and a Windows/NT version will follow. This will provide an opportunity to test Mariposa on heterogeneous architectures. To complement this work, we would also like to take advantage of Mariposa's bid curve to assign queries to different execution classes, such as "fast and expensive" and "slow and cheap" and support the appropriate execution of such classes.

Data placement can have a profound impact on database performance. Mariposa supports lightweight data movement, copies and data fragmentation. The Mariposa data broker [STO96] buys and sells database tables or copies of tables in much the same way that the query broker buys and sells other computational resources. It is our belief that simple data broker algorithms, much like the ones presented in this paper, can lead to effective data placement.

Acknowledgments

I would like to thank the following people: Mike Stonebraker, for his guidance and support; Andrew MacBride, our chief programmer, for helping me run the benchmarks; Joe Hellerstein for his input on, and criticisms of, this paper; the rest of the Mariposa group for their suggestions and comments (and the use of their machines).

References

[BER81] P. A. Bernstein, N. Goodman, E. Wong, C.L. Reeve, J. Rothnie "Query Processing in a System for Distributed Databases (SDD-1)," ACM Transactions on database Systems, 6(4), (December, 1981).

[BIT83] D. Bitton, et al "Benchmarking Database Systems: A Systematic Approach," Proc. 1983 VLDB Conference, Florence, Italy, Nov. 1983.

[CHM95] C. Chekuri, W. Hasan, R. Motwani, "Scheduling Problems in Parallel Query Optimization," Proceedings of the Fourteenth ACM Symposium on Principles of Database Systems (PODS), 1995, pp. 255-265.

[DAV95] D. Davison, et al. "Dynamic Resource Brokering for Multi-User Query Execution," Proceedings of the 1995 ACM SIGMOD (May, 1995), pp. 281-92.

[DNS91] D.J. DeWitt, J.F. Naughton, D.A. Schneider, S. Seshadri, "Parallel Sorting on a Shared-Nothing Architecture Using Probabilistic Splitting." Proc. of the First International Conference on Parallel and Distributed Information Systems, Miami, Florida (December 1991), pp. 280-291.

[DNS92] D.J. DeWitt, J.F. Naughton, D.A. Schneider, S. Seshadri, "Practical Skew Handling in Parallel Joins," Proc. 18th VLDB Conf. (1992), pp. 27-40.

[GRA93] J. Gray, G. Graefe, "Transaction Processing: Concepts and Techniques," Morgan Kaufmann Publishers, Inc. San Mateo, CA, ©1993.

[HAS95] W. Hasan, "Optimizing Response Time of Relational Queries by Exploiting Parallel Execution," PhD thesis, Stanford University, 1995. In preparation.

[HLY93] K.A. Hua, Y. Lo, H.C. Young, "Considering Data Skew Factor in Multi-Way Join Query Optimization for Parallel Execution," VLDB Journal 2(3) (1993), pp. 303-330.

[HM95] W. Hasan, R. Motwani, "Coloring Away Communication in Parallel Query Optimization," 1995. Submitted for publication.

[HON92] W. Hong. "Parallel Query Processing Using Shared Memory Multiprocessors and Disk Arrays," PhD thesis, University of California, Berkeley, August 1992.

[IOA92] Y. Ioannidis, et al. "Parametric Query Optimization," Proceedings of the 18th International Conference on Very Large Databases (August, 1992)

[MAR96] "The Mariposa User's Guide," http://mariposa.berkeley.edu (1996).

[MD95] M. Mehta, D.J. DeWitt, "Managing Intra-Operator Parallelism in Parallel Database Systems," Proc. 21st VLDB Conf., (1995) pp. 382-394.

[OUS90] J. Ousterhout. "Tcl: An Embeddable Command Language," Proceedings of the Winter 1990 USENIX Conference (January, 1990), pp. 133-46.

[RM95] E. Rahm, R. Marek, "Dynamic Multi-Resource Load Balancing in Parallel Database Systems," Proceedings of the 21st VLDB Conf (1995), pp. 395-406.

[SEL79] P. Selinger, et al. "Access Path Selection in a Relational Database Management System," Proceedings of the 1979 ACM SIGMOD Conference on Management of Data (June, 1979).

[SID96] J. Sidell, et al. "Data Replication in Mariposa," Proceedings of the 12th International Conference on Data Engineering (February, 1996).

[STO86] M. Stonebraker "The Design and Implementation of Distributed INGRES," in The INGRES Papers, M. Stonebraker (ed.), Addison-Wesley, Reading, MA, 1986.

[STO96] M. Stonebraker, et al. "Mariposa: A Wide-Area Distributed Database System," VLDB Journal 5, 1 (Jan. 1996), pp. 48-63.

[TPC] Transaction Processing Council, 777 N. First St. Suite 600, San Jose, CA 95112-6311. URL: www.tpc.org

[WDH81] R. Williams, D. Daniels, L. Haas, et al, “R*: An Overview of the Architecture,” IBM Research, San Jose, CA, RJ3325, Dec. 1981.

[WDJ91] C.B. Walton, A.G. Dale, R.M. Jenevein, “A Taxonomy and Performance Model of Data Skew Effects in Parallel Joins,” Proc. 17th VLDB Conf. (1991) pp. 537-548.

[WDY91] J.L. Wolf, D.M. Dias, P.S. Yu, J. Turek, “An Effective Algorithm for Parallelizing Hash Joins in the Presence of Data Skew,” Proc. 7th IEEE Data Engineering Conf. (1991), pp. 200-209.

[ZG90] H. Zeller, J. Gray, “An Adaptive hash Join Algorithm for Multiuser Environment,” Proc. 16th VLDB Conf. (199), pp. 186-197.