# The Mariposa Distributed Database Management System

Jeff Sidell
jsidell@cs.berkeley.edu

## Introduction

The Mariposa distributed database management system is an ongoing research project at the University of California at Berkeley under Professor Michael Stonebraker. This project addresses several issues in distributed data management and has defined the following goals:

*Scalability*: Our goal is for Mariposa to scale to 10,000 sites. This makes it necessary for Mariposa sites to operate autonomously and without global synchronization. Database activities such as class creation, updates, deletions, fragmentation and data movement must happen without notifying any central authority.

*Fragmentation*: Every Mariposa table, or class, is horizontally partitioned into a collection of **fragments** which together store the instances of the table. The collection of fragments can be **structured** (partitioned by predicate-based distribution criteria) or **unstructured** (partitioned randomly or in round-robin order).

*Data movement*: Fragments can move from one site to another without quiescing the database. Data movement makes it possible for Mariposa sites to offload and obtain data objects, resulting in load balancing and better system throughput.

*Flexible copies*: Copies can enhance data availability and provide faster query processing through parallel execution. However, the cost of maintaining the consistency of a large set of replicas can be prohibitive if conventional techniques (e.g., two-phase commit) are used. Mariposa provides a replica-management system that avoids the expensive synchronization requirements of conventional replica systems without sacrificing transaction serializability. Copies are at the granularity of fragments. The copy mechanism is discussed at greater length in [3].

*Flexible system management*: The behavior of Mariposa sites is controlled by scripts written in an enhanced version of Tcl.

Mariposa is an example of an **agoric** system. The term agoric comes from the Greek word for market place: *agora*. An agoric system mimics a capitalist economy with buyers negotiating with sellers to purchase goods and services. In Mariposa, the buyers and sellers are the Mariposa servers. The goods are computational resources (CPU time, disk space and bandwidth, and network capacity) and data. Mariposa sites negotiate with one another to perform query processing, move data between sites and so forth. Agoric systems have the key feature of extreme scalability. System information is decentralized: there is no master site.

## The Mariposa Architecture

Mariposa consists of the modules shown in Figure 1. The user submits a query such as `SELECT * FROM EMP` 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 unit of cost is dollars. The home site is simply the site where the query originated. The query passes through a parser and an optimizer and is turned into a query plan. In Figure 1, the query `SELECT * FROM EMP` has been transformed into a scan over the EMP relation. The optimizer can be as simple or as sophisticated as necessary. Both the parser and the optimizer (and, later, the query broker) use information from a Mariposa *name server*. Name servers provide system metadata including type information, data fragmentation and placement, etc. 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. In Figure 1 the scan of the `EMP` relation has been transformed into two scans over the fragments `EMP1` and `EMP2`, which are merged together.

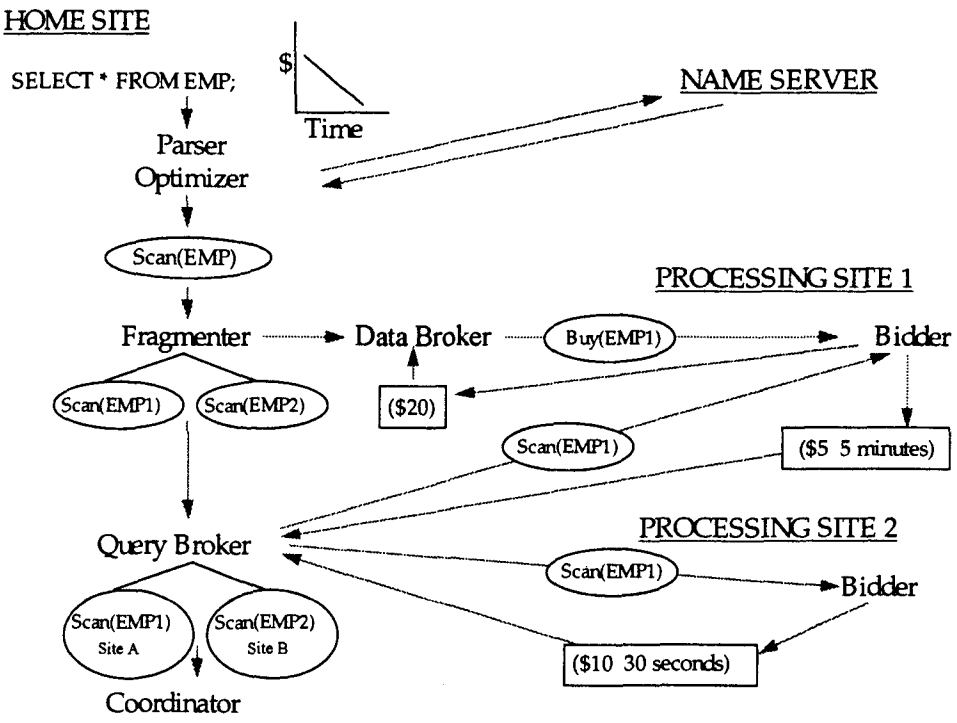


**Figure 1**

The fragmented plan is passed into the *query broker*, whose job it is to assign a processing site to each node in the plan tree. The query broker follows one of two protocols: In the *long protocol*, illustrated in Figure 1, the query broker contacts *bidder* processes running at potential processing sites, passing along some portion of the query plan and soliciting a bid. The 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 scan over `EMP1`, and gotten back two bids: `($5, 5 minutes)` and `($10, 30 seconds)`. The bidder's response is determined completely by a script written in Tcl, to which we have added extensions. The broker selects the set of bids that will solve the query as far under the bid curve as possible. 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. The user's bid curve will determine in part the sites that will process the query. For example, a Cray would probably charge more than a PC but take less time. A bid curve with a steep slope, indicating that a fast answer is valuable, would select the faster processor.

In the *short protocol*, which is not shown in Figure 1, the broker does not contact bidder sites first, but uses information gathered from the name server and from previous queries to select processing sites. It then contacts the processing sites, telling them to run the subquery and send a bill. The short protocol can result in a query running for a cost and time that are more than the user specified in the bid curve. It can also result in a query not being run at all, since the bidder sites can refuse to process the subquery sent by the broker.

There is also a *data broker* process running at each site. The data broker determines which data fragments the site will attempt to acquire and which fragments the site will get rid of. It uses information about which data fragments have been accessed in the past and how much has been spent to access them. The data broker's behavior, like that of the bidder, is determined by a script written in an extended version of Tcl. The data broker also contacts bidder sites to obtain the purchase price for a fragment it is interested in buying, or the sale price for a fragment it is interested in selling. In

Figure 1, the data broker has asked Site 1 how much it would cost to buy the **EMP1** fragment. Site 1 has sent back a selling price of $20.

The Mariposa architecture and the economic paradigm are discussed further in [1], [2] and [4].

## The Fragmenter

After the static optimizer passes the query plan to the fragmenter, the fragmenter breaks up the underlying table scans into fragmented scans. If there is more than one relation involved in the query, and if at least one of the relations is fragmented, there is more than one fragmented plan that is equivalent to the original unfragmented plan. For example, the query "**SELECT * from EMP, DEPT where EMP.deptno = DEPT.no**" may be converted into the plan tree shown in Figure 2 by the optimizer before fragmentation. In this plan, the two relations, **EMP** and **DEPT**, are first scanned (denoted by **SS**) and then sorted into temporary relations before being joined. The join algorithm used in this example would be a merge join.

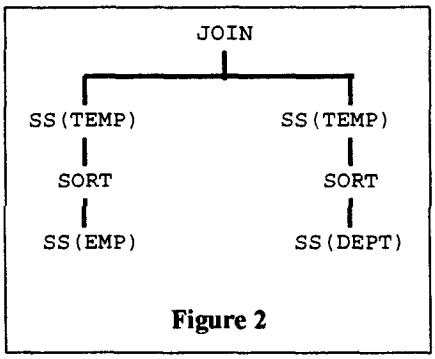


**Figure 2**

Assume the **EMP** relation were partitioned into two fragments, **EMP1** and **EMP2**, and the **DEPT** relation were fragmented similarly into **DEPT1** and **DEPT2**. The fragmenter could produce either of the plans shown in Figure 3, among others. The placement of *merge nodes* in the plans shown in Figure 3 affects their potential for parallel execution. Merge nodes accept multiple tuple streams as input and output a single tuple stream. In the top plan in Figure 3, the merge nodes have been placed immediately above the sequential scans, essentially turning the fragmented scans into scans over the entire relation. The rest of the plan is identical to that in Figure 2. In the bottom plan, the merge node was inserted at the root of the plan tree. Each storage fragment is sorted and joined with each fragment from the other class, and the results are merged. In general, if relations **A** and **B** are divided into $f_A$ and $f_B$ fragments, a join over **A** and **B** can be divided into as few as one or as many as $f_A \times f_B$ joins. The greater the number of joins, the greater the potential for parallel execution of the plan.

## The Query Broker

The Mariposa query broker, after being handed the fragmented query plan by the fragmenter, assigns a processing site to each node in the query plan using either the short protocol or the long protocol. Using the long protocol assures that the query can be run within the user's time/cost constraints at the processing sites which have won the bidding process. The long protocol also allows the system as a whole to adapt dynamically to changing resource utilization. The processing sites may adjust their bids according to the current demand for resources. This use of "market forces" results in a natural form of load-balancing: as resources become overutilized, or scarce, their prices can be raised and less expensive alternatives are used. In the long protocol, after the query broker accepts the fragmented plan from the fragmenter, the first thing it does is to divide the fragmented plan into a set of non-overlapping subplans, called *plan chunks*. Each plan chunk is sent out whole to potential processing sites. 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. The effects on query processing of breaking up plans in different ways is an area for future study.

The second thing the query broker does is determine the set of bidder sites to contact for each plan chunk. The set of bidder sites contacted is determined by a Tcl script, and can therefore be changed by the user. After determining the bidder sites for each plan chunk, the query broker solicits each site for a bid, passing it the plan chunk. When the query broker has received bids from all the bidder sites, it determines the set of bidders which can

run the query as far below the bid curve as possible. Control is then passed to the coordinator, which notifies the winning bidders and coordinates execution of the final plan.

## The Bidder

The bidder site receives a plan chunk from a remote query broker and passes it to the bidder script. The bidder script returns a value indicating whether or not it is willing to bid, and if it is willing, it also returns the cost and time estimate. The bidder script is written in an enhanced Tcl. The query plan passed in to the bidder is represented as a Tcl list. Information about the base relations accessed by the query, such as number of tuples and number of pages, is also made available to the Tcl script. The bidder may formulate its bid in any way. As an example, the bidder could recursively descend the query plan, assigning a cost to each node depending on the estimated resource consumption (disk I/O and CPU time), effectively mimicking a traditional static single-site optimizer. This pricing strategy could be enhanced by having the bidder multiply the cost-based bid by the current load average. This would cause busier sites to charge more than idle sites, providing a crude form of load-balancing.

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 2 were received by a bidder site which only had the EMP relation but not the DEPT relation, it could process all of the plan except the scan over DEPT. In this case, the bidder can solicit sub-bids from other bidder sites for the part of the plan it cannot process itself. It adds the sub-bids into the bid it returns to the original broker site.

## The Data Broker

The data broker's behavior, like that of the bidder, is determined completely by scripts written in an enhanced Tcl. The data broker may make use of system catalogs that record the sub-plans which originated locally and how much was paid to process them. We have added a command to Tcl which allows the data broker to query these catalogs: the **CostHistory** command takes two arguments: a list of operations the broker is interested in, and a time interval. The data broker queries the system

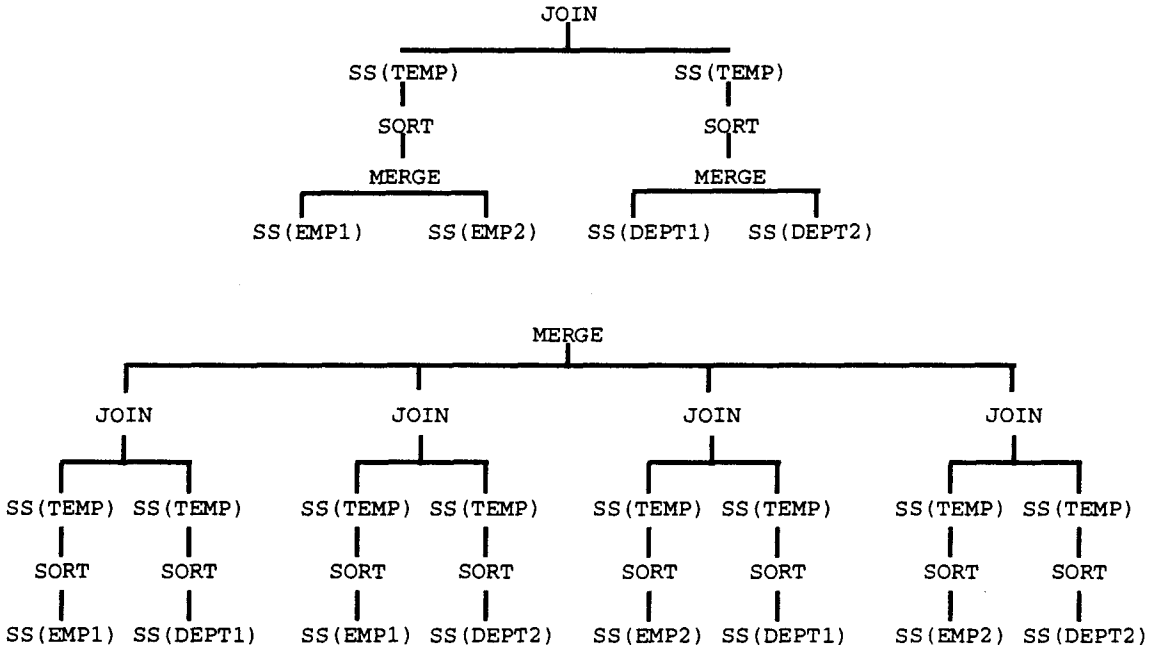


**Figure 3**

catalogs for subplans which contain all of the operations and were processed within the time interval. For example, to find out how much the site has paid for joins between R1 and R2 during the last hour, the data broker would make the call "`set Cost [CostHistory "R1 R2 JOIN" "NOW-1 NOW"]`".

We have also added the **PurchaseCost** command. PurchaseCost takes a fragment name as an argument and returns a list of storage sites and how much they will charge to sell the fragment or a copy of the fragment. For example, the return value for **PurchaseCost EMP1** may be **(1 5) (2 10)**, indicating that EMP1 is stored at sites 1 and 2, and that they will charge $5 and $10 respectively to sell it. In order to buy a fragment, the data broker uses the Tcl extension **PurchaseFragment**, which takes the fragment name and the storage site from which the data broker will purchase it.

## Implementation Status

The Mariposa system was released in June of this year and is currently ported to Digital Equipment's Alpha architecture running OSF/1. We are porting to other architectures and expect to have versions for Solaris and Windows/NT by the first quarter of 1997. The Mariposa source code, documentation and papers are available from the Mariposa web server at http://epoch.cs.berkeley.edu:8000/mariposa.

The author is currently conducting experiments to test the system's ability to perform automatic load balancing through the query, automatic data layout through the data broker, as well as the system's ability to scale.


[1] *Mariposa: A New Architecture for Distributed Data*, Sequoia 2000 Technical Report 93/31, University of California, Berkeley, CA, May 1993. Appeared in: *Proceedings of 1994 IEEE 10th International Conference on Data Engineering*, Houston, TX, USA, 14-18 Feb. 1994. Los Alamitos, CA, USA: IEEE Comput. Soc. Press, 1994. p. 54-65.

[2] *An Economic Paradigm for Query Processing and Data Migration in Mariposa*, Sequoia 2000 Technical Report 94/49, University of California, Berkeley, CA, Apr. 1994. Appeared in: *Proceedings of 3rd International Conference on Parallel and Distributed Information Systems*, Austin, TX, USA, 28-30 Sept. 1994. Los Alamitos, CA, USA: IEEE Comput. Soc. Press, 1994. p. 58-67.

[3] *Data Replication in Mariposa*, Sequoia 2000 Technical Report 95/62, University of California, Berkeley, CA, June 1995. Appeared in: *Proceedings of the 12th International Conference on Data Engineering*, New Orleans, LA, Feb. 1996.

[4] *Mariposa: A Wide-Area Distributed Database System*, Sequoia 2000 Technical Report 95/63, University of California, Berkeley, CA, June 1995. Appeared in: *VLDB Journal* 5, 1 (Jan. 1996), p. 48-63.