The Network is the Database:

Data Management for Highly Distributed Systems

Julio C. Navas
Siemens Technology-to-Business Center
1995 University Ave, Suite 375
Berkeley, California, 94704
julio@ttb.siemens.com

Michael Wynblatt
Siemens Technology-to-Business Center
1995 University Ave, Suite 375
Berkeley, California, 94704
wynblatt@ttb.siemens.com

ABSTRACT

This paper describes the methodology and implementation of a data management system for highly distributed systems, which was built to solve the scalability and reliability problems faced in a wide area postal logistics application developed at Siemens. The core of the approach is to borrow from Internet routing protocols, and their proven scalability and robustness, to build a network-embedded dynamic database index, and to augment schema definition to optimize the use of this index. The system was developed with an eye toward future applications in the area of sensor networks.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems – *distributed databases, query processing.* H.4.4 [**Information Systems Applications**]: Miscellaneous. C.2.4 [**Computer-Communication Networks**]: Distributed Systems – *distributed applications, distributed databases*.

General Terms

Algorithms, Performance, Design.

Keywords

Wide-Area Data Management, Sensor Networks, Logistics, Distributed Data Management.

1. BACKGROUND & MOTIVATION

The rapid decrease in the cost and size of data communications hardware and various sensor technologies offers to support a wealth of new businesses, loosely termed "sensor networks" [3,5]. Applications have been proposed in areas including intelligent highways, power grid management, intelligent battlefields, and remote product service and maintenance. Although the applications are varied, they share several common features: (1) a relatively large number of data sources (typically on the order of 10^5 or more), (2) relatively volatile data and data organization, and (3) and the requirement for "thin" data servers, to run on scaled down hardware.

ACM SIGMOD 2001 May 21-24, Santa Barbara, California USA

These applications offer a significant challenge for data management. Most application-level tools wish to treat such a collection of data as a traditional database, using well-known query languages to access the data. In the state-of-the-art of commercial systems, the solution to similar problems is to use a centralized database (or small collection of distributed databases), and collect the data as fast as possible, either through polling or event-driven reporting. Applications then act on the database in the traditional way.

However, in applications requiring tens of thousands of data sources, with rapidly changing data, such systems are insufficient. The databases themselves, and especially their data communications channels, become bottlenecks that prevent the system from achieving these scales. Moreover, such databases represent critical failure points, and also introduce latency that may be relevant in real-time applications. A system is needed which is highly scalable, offers no critical failure points, and lets data flow from the source to the requestor as rapidly as possible. One research project addressing this space is COUGAR [1], but this system assumes a centralized index of all data sources, which does not address our scalability or critical failure point requirements.

At Siemens, we see a great future potential in sensor networks applications, and in developing technology that will satisfy these very challenging constraints. More immediately, we have a need to support a related application in the area of postal logistics. This application has somewhat milder constraints (less than one hundred thousand data sources, and reasonably fat clients are acceptable), but offers many of the same challenges as the sensor networks. Our goal was to produce a solution that solved our immediate problem and has applicability to the more demanding problems we foresee in the near-term.

The postal logistics application will serve as a guide for this paper, as we show examples from this domain to illustrate how our system operates. Section 2 describes this application. Section 3 describes a wide area data management system, and details specific to the implementation are described in Section 4. Our future work is described in Section 5.

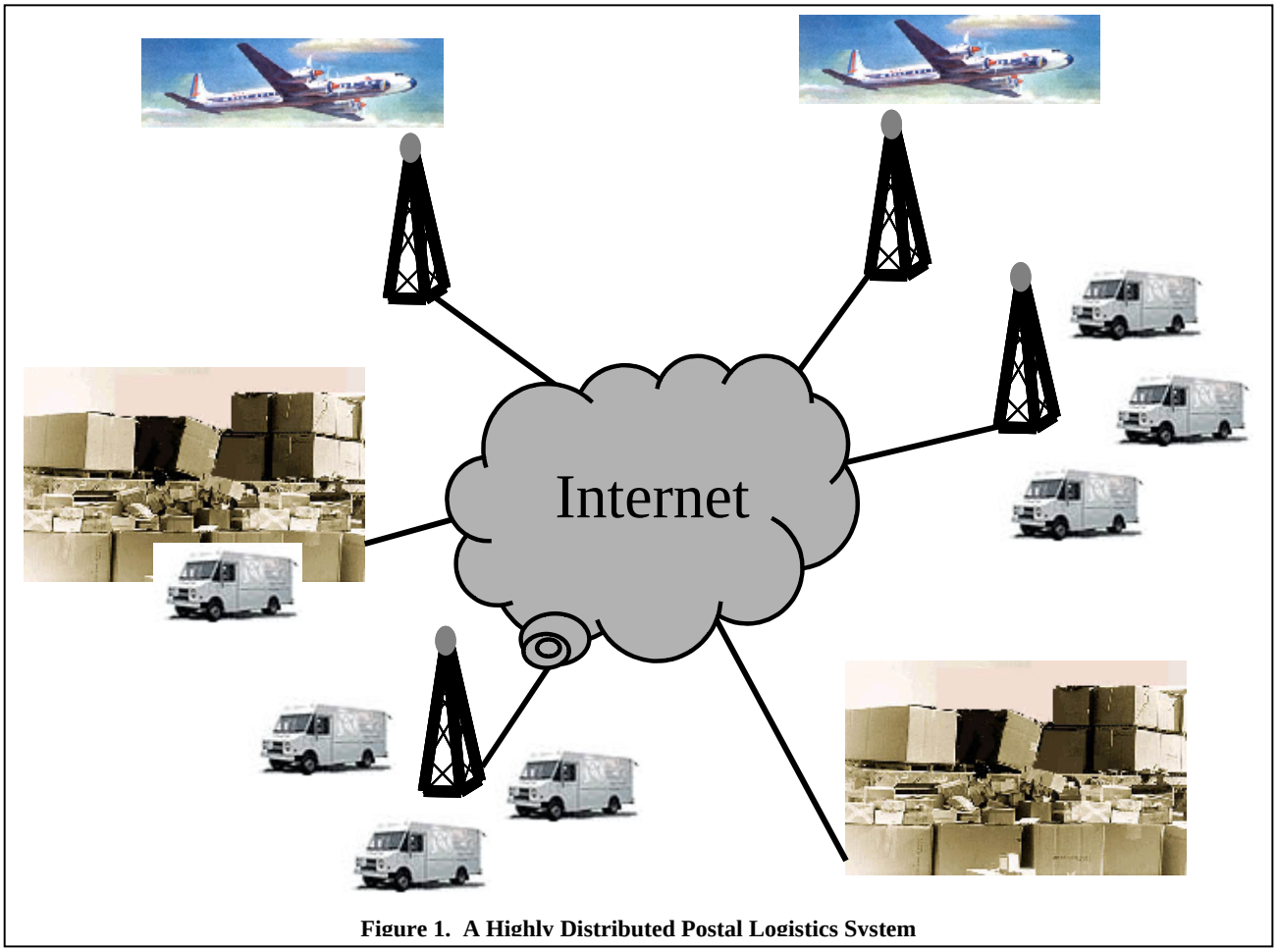

Figure 1. A Highly Distributed Postal Logistics System

2. DATA MANAGEMENT IN WIDE AREA POSTAL LOGISTICS

Figure 1. shows an overview of the application that we support. The idea is to allow a large courier service to treat its distribution and staging system of hubs, substations, trucks, and airplanes as a single live database of packages. Any authenticated client, from any point in the network, can issue a query against this database and receive the current answer.

A typical scenario for this system is the following. A logistics agent (automated or human) located at an airport hub is faced with a partially loaded airplane scheduled to depart shortly. An optimization question arises: should the plane embark partially loaded, or await the arrival of additional packages? The agent issues a query to ask how many packages bound for the airplane's destination, and in what sizes, are expected to arrive at the airport within the next hour. In order to answer this correctly, the system must have access to the status and contents of all vehicles currently bound for the airport.

Requirements for this application include:

- On the order of 50,000 data sources which are so-called "tertiary containers" (trucks, airplanes, stations, etc.).
- Data sources may autonomously add or remove themselves from the system at any time.
- Data currency to within 5 minutes of real-time.
- Queries should be submitted in a SQL-like language familiar to application programmers.
- Never have a total system failure, and minimize the impact of partial failures.

In addition to these constraints, we were also concerned with the processing capability required at the data sources. Although it is reasonable for tertiary containers to include a PC-like device, we have our eye on future sensor network applications, which require thinner servers.

For the purpose of our work, we assume that tertiary containers are appropriately updated with the information regarding the packages they currently hold. Electronic information about a package moves with the package. This is achieved using RF-ID tagging and other techniques, and is beyond the scope of this paper.

Figure 2 shows a simplified data schema that might be used for this application. Each table is distributed, storing the data along with its physical manifestation. In this example, Vehicle and Station records are stored at the location of the vehicle or station.

Package records are stored at the site of the package, either vehicle or station. Customer records are associated with packages, and are stored with them. `ConveyedBy` and `StoredAt` tables are stored with the corresponding vehicle or station.

```
Package(PID, Size, SenderID, ReceiverID,
DestStation, DestZIP, Priority,
SpecHandling)
Vehicle(VID, Dest, ExpectedWait, Status,
VType)
Customer(CID, Name, StreetAddr, City,
ZIP)
ConveyedBy(PID, VID)
Station(SID, Name, State, Country,
Region)
StoredAt(PID, SID)
```

Figure 2. A simplified schema for the postal logistics Application

3. THE NETABASE APPROACH

Our approach, called Netabase, is fundamentally based on the observation that the Internet scales very well. The IP routing protocol which underlies the Internet supports millions of nodes, and allows significant dynamism and autonomy for nodes to add or remove themselves.

We have developed a new network routing technology called *characteristic routing*, to support the scalable and efficient routing of data queries to data sources which have relevant information. In effect, the computer network becomes a distributed, dynamic index to the data sources. We define a methodology to determine appropriate keys for this index, with the principal goal to minimize network traffic. At a node issuing a query, a system for query decomposition uses these keys to send the components of the query to just the right places, and receives and integrates the replies.

Figure 3 shows an overview of the system. An application issues a query in our SQL-like language. The query decomposer breaks the query into components deliverable to different types of data sources, and chooses appropriate "routing keys". The characteristic virtual routers route the query fragments to just these data sources. The thin servers at the data sources evaluate the query fragments, and reply. The replies are integrated at the requesting node.

Not shown in the figure are the system's administrative tasks. First, we have extended the schema definition specification of our SQL-like language to include optimization specifications. Second, the characteristic routers continuously communicate with each other to update their routing tables to reflect changes in the distribution of characteristics.

3.1 Local Join Specification

An important consideration in reducing data traffic is the possibility to perform some join operations at the data sources. In general, if data is provided from several sources, joins cannot be performed at the sources, due to the possibility of filtering out records that would correctly join with records from other sources. However, the semantics of a particular schema may indicate that certain joins can be performed at the sources without fear of loss of information.

In our example schema, the `Vehicle` and `ConveyedBy` tables are an example of a pair of locally joinable tables. The semantics of the system dictate that all records of a `ConveyedBy` table that correspond to a particular `Vehicle`.VID are located at the same data source (the vehicle) as the `Vehicle` record for that `Vehicle.VID`. If a join is requested on these two tables, the join can be performed at the vehicle data source, without fear that records will be lost which might be joinable elsewhere.

It is further possible that certain attributes of a pair of tables will be locally joinable, even if not all attributes of those tables are locally joinable. In our postal schema, for example, `Package.ReceiverID` is locally joinable with `Customer.CID`, because a `Customer` record describing the receiver of a package travels with each package. However, `Package.DestZIP` is not locally joinable with `Customer.ZIP`. To see this, consider a query that tries to find the names of all customers who live in the same ZIP that a package is destined for. This query would typically involve a join on the `ZIP` attributes. There may be many customers who live in this ZIP code, and they will not all be recorded in a given truck. Thus a join on the ZIP attribute cannot be conducted locally, or

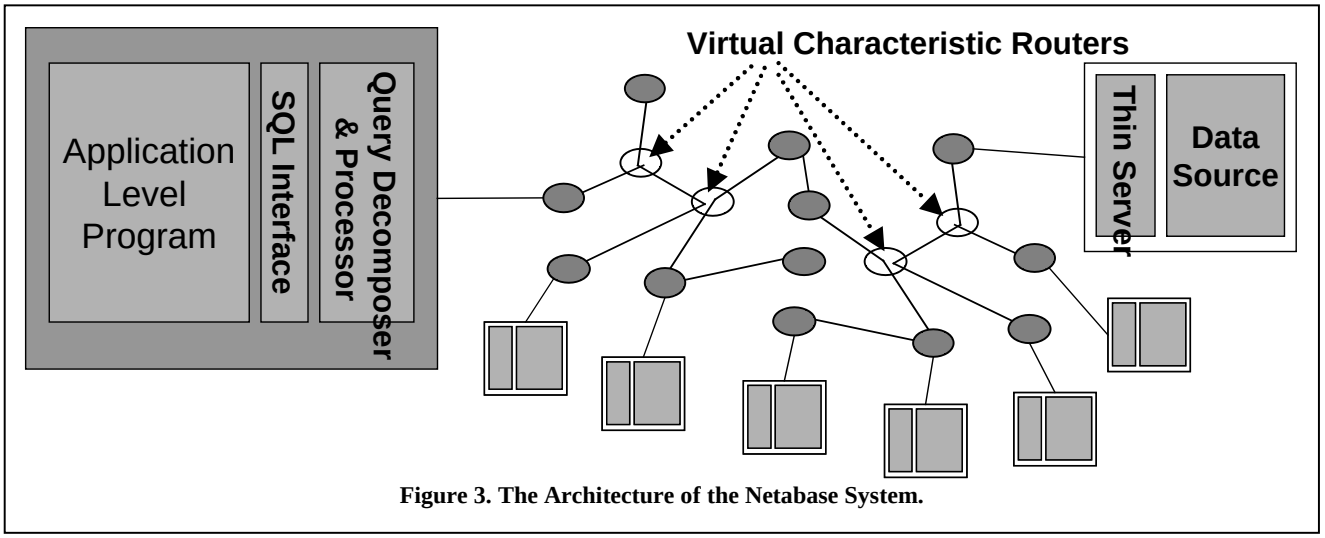

Figure 3. The Architecture of the Netabase System.

many potential matches would be overlooked.

In order to allow optimization of a query based on local joins, we have extended the specification of the schema to include local join relationships. All joins are assumed not to be locally joinable unless identified through the use of this extension. The syntax allows either pairs of complete tables, or specific pairs of table attributes to be designated as locally joinable.

```
join_locally_statement-> join_locally table, table
  | join_locally table.attribute, table.attribute;
```

3.2 Query Decomposition

The goal of the query decomposition stage is to divide the query into pieces so that data sources receive only the parts of the query which are relevant to them, and which are independently processable by them. For example, a data source should not receive a request for information about a table in which it does not take part. Moreover, if one or more operations in a query, such as a join, are not processable locally at a data source, then some pieces of the same query may need to be satisfied separately. Each resulting piece can be considered as a message, to be sent to the appropriate data sources as a request for information.

Additionally, the messages into which the query is divided should be satisfied in an order which is consistent with the goal of minimizing network traffic.

To achieve these goals, the query decomposer's task can be roughly divided into three sub-tasks: (1) division of the query into predicate groups based on query structure, (2) further division of the predicate groups based on relevant tables and possibilities for local joins, and (3) ordering of the resulting groups.

The first step is to divide the query into predicate groups. The general approach is illuminated by the consideration that `AND` operations in a WHERE clause act as filters reducing the number of satisfying records, while `OR` operations indicate independent clauses. For that reason it is useful to group `AND`ed predicates into the same message where possible, to reduce the number of replies, and `OR`ed clauses are separated and handled independently. The `WHERE` clause of each sub-query is converted to disjunctive normal form to facilitate this division, and subqueries are treated independently.

After predicate groups have been established based on separating the `OR` clauses in each subquery, they are refined by further dividing them into local-join groups. Local-join groups are groups of predicates in which all of the predicates can be resolved locally at the same data source. Local join groups are computed by first clustering all non-join predicates by the table they reference, and then unifying groups through local joins which are specified. Non-local join predicates are then included in any local-join group which they reference, which means they will appear in multiple groups. Figure 4 shows a query and the resulting predicate groups, assuming that the `Package` and ConveyedBy, and `Vehicle` and `ConveyedBy` tables have been specified as locally joinable.

```
SELECT P.Size
FROM   Vehicle V, Package P,
       ConveyedBy CB
WHERE  V.VID = CB.VID
  AND  P.PID = CB.PID
  AND  P.DestHub = "Chicago"
  AND  V.Dest = "LA Airport"
  AND  V.ExpectedWait < 60
```

Non-joins Grouped By Table:

Group 1: `P.DestHub = "Chicago"`

Group 2: `V.Dest = "LA Airport"`
`V.ExpectedWait < 60`

Unified through Local Joins

Group 1: All 5 predicates

Figure 4. An example query and predicate groups

Next, the predicate groups within each subquery or `OR` clause are ordered. The discussion of how the ordering is done is deferred to section 3.4 on query evaluation, in order to give the proper context. Predicate groups in different subqueries or `OR` clauses can be resolved independently, and thus are issued in parallel. When a predicate group includes an `IN` predicate (indicating a subquery), that group must wait for the subquery (possibly several predicate groups) to be resolved before it can be resolved itself.

Additional optimization is possible based on elimination of common sub-expressions, but we did not implement this.

3.3 Characteristic Routing

Messages are sent to data sources over the Internet using a new message routing protocol called characteristic routing. In our prototype Netabase system, characteristic routing is implemented as a virtual routing protocol on top of IP, but it can also be implemented in place of IP.

Using the characteristic routing protocol, internal nodes in the network route messages to data sources which have particular characteristics. A single message can be sent to multiple destinations, as characteristic routing uses techniques derived from IP-multicast [2] and related to geographic routing [6]. Although characteristic routing allows arbitrary strings to be used as characteristics, we choose special strings which serve as an index to the database. Since the routing tables are constantly updated, the database index is highly dynamic.

The networking aspects of characteristic routing will be addressed in another paper. In this paper we focus on the manner in which characteristic routing supports the wide area database.

3.3.1 Routing Keys

Our approach is based on the observation that different data sources participate in different database tables. As was described in section 2, our postal application maintains data stored at

vehicles and stations, with the former participating in vehicle relevant tables, the latter participating in station relevant tables, and both participating in package tables. In sensor networks, different sensor types typically represent different tables. For example in a chemical processing plant, temperature, pressure, current volume and valve status are tables participated in by the sensors which measure these variables.

As a first pass, each data source advertises as characteristics the names of the tables in which it participates in. Whenever a message is issued, the tables involved in that message are specified as the "routing keys" for the message, and the message will be routed only to those data sources which advertise those characteristics.

More accurate routing can be achieved by using a combination of attribute names and values as a routing key. For example, if the predicate group from figure 4 is issued as a message, instead of using ("`Vehicle`", "`Package`", "`ConveyedBy`")as the routing key, one could more accurately route the message by using "`Vehicle.Dest=LA_Airport`". Using the first set of characteristics, all vehicles would receive that message, but only a small subset of vehicles, topologically clustered, would receive the message using the second characteristic.

The trade-off for the accuracy gained through value-based routing keys is larger routing tables and additional administrative traffic. Since there are many more possible attribute values than there are table names in a database, allowing values to be used as routing keys increases the size of the routing table dramatically. This can be offset by only maintaining routing information for attributes with high ranks. Another approach to managing the size of the routing tables is to abstract the information in the routing table hierarchically. This approach is described in section 3.3.3. In addition, since attribute values change more frequently than table names, more administrative overhead is needed to maintain the routing tables. Therefore, the use of values as routing keys is not useful for attributes where the value typically changes at a frequency comparable to the latency of the system. The use of hierarchical abstraction limits the proliferation of updates through the network, and is thus doubly valuable.

3.3.2 Building the Routing Tables

Data sources advertise their characteristics to their local characteristic router. In our initial implementation, data sources also advertise deletion of characteristics that had been previously advertised. We believe a significant reduction in traffic can be achieved with a soft-state approach, in which characteristics that are not re-advertised grow stale after a timeout period and are discarded. This approach would also directly support data sources which remove themselves from the system.

Within the network, characteristic routers continuously exchange routing instructions for the characteristics they are aware of. This provides the wide area index for Netabase.

3.3.3 Routing Key Abstraction

In order to maintain the characteristic routing tables at a reasonable size, abstraction techniques are used in order to "throw-away" most of the information. Whereas a characteristic can be any arbitrary string of arbitrary size, a characteristic router only needs to know that a particular characteristic exists and that a particular destination in the network is associated with that characteristic. This observation allows us to ignore the actual contents of the characteristic itself and simply record its presence (or not) for a particular destination using bit vector techniques that were pioneered in Information Retrieval [4]. In this manner, each destination in the network will have a bit vector associated with it in which each of the bits corresponding to the characteristics belonging to that destination will be "on."

In order to achieve even greater savings in routing table size, the bit vectors themselves are further abstracted. This abstraction takes the form of associating a single bit with a range of characteristics. The range of characteristics corresponding to each bit can be varied with a corresponding smaller decrease in the routing accuracy. This allows the routing table size and the control message overhead to be potentially greatly reduced and, thereby, making the system more amenable for use in lower-bandwidth networks such as wireless networks. However, this raises the possibility that some data sources will receive messages not intended for them. Where bandwidth is more plentiful, the range of characteristics corresponding to each bit can be decreased leading to greater routing accuracy with a corresponding increase in memory and control overhead requirements. This trade-off introduced is similar to the bandwidth/memory trade-off described in [9].

The amount of information accuracy that is kept per destination is also varied with the network distance to that destination. More accurate information is kept about destinations that are "nearby" versus destinations that are farther away. Essentially, the amount of bit-vector abstraction is increased for distant destinations. The intuition underlying this can be illustrated with the following example: A characteristic router in California receives a packet bound for a set of characteristics that belong to a destination in New Jersey. The California router, however, does not need to have detailed knowledge about the network structure within New Jersey or even detailed knowledge about the exact characteristics found in New Jersey. All it needs to know is that a destination with characteristics "similar" to the packet's destination characteristics can be found to the "east." Therefore, the router will forward the packet eastward. As the packet nears its eventual destination, the information accuracy increases so that the packet finds its way correctly to the proper final destinations.

3.3.4 Choosing a Routing Key

For any given message, a routing key is derived from its predicate group. Using the ranks of the predicates described in section 3.4.2, the highest ranked non-join predicate is selected, and the routing key is created of the form `TABLE_ATTRIBUTE_VALUE`, although we use truncation to reduce string lengths. In the case that either there are no non-join predicates, or that there are no non-join predicates which include routable attributes, the set of table names referenced in the predicate group is used as routing characteristics.

3.4 Query Evaluation

3.4.1 Unfolding Non-Local Joins

Non-local joins are any joins which have not been specified to be resolved locally. This means that the records to be joined must come from multiple data sources. Rather than gathering all of the data and then joining it, we choose to "unfold" the joins, that is to

gather the records from one table, and use that to filter the data coming back from the other table. This approach is similar to the traditional semi-join algorithm for distributed joins [8], with the addition of a selection in the first step which reduces the number of records transmitted.

```
SELECT V.VID, S.Name
FROM   Vehicle V, Station S
WHERE  V.Dest = S.Name
  AND  S.State = "California"
  AND  V.ExpectedWait < 60

_____________________________________________

Group 1:  S.State = "California"
          V.Dest = S.Name
Group 2:  V.ExpectedWait < 60
          V.Dest = S.Name
```

Figure 5. An example query and predicate groups

Figure 5 shows an example of a query with a non-local join, and the predicate groups generated. Note that the join predicate appears in both groups, since it involves both tables. When this query is resolved, the join is unfolded in the following way. When message 1 is sent, the join predicate is removed, and `S.Name` is requested as a return value. The `S.Name` data is collected, and is included in message 2. Data sources only reply to message 2 if they have data which will satisfy the join predicate given the included data. Unfolding the joins in this manner acts as a very coarse filter on replies, and significantly reduces the network traffic generated as a result of the query.

Figure 6 shows the possible text of message 2, where `%1` indicates a placeholder for which the data listed is to be substituted. Data sources interpret this locally like a logical `OR` of four predicates.

```
Vehicle.ExpectedWait < 60
Vehicle.Dest = %1;
(%1,String)"LosAngeles", "Oakland", "Hayward",
"Santa Barbara";
Vehicle.VID
Vehicle.Dest;
```

Figure 6: A Message from an Unfolded Join.

3.4.2 Ordering Predicate Groups

After predicate groups have been determined, it remains to be decided in what order they will be sent as messages. Choosing the right order can have a significant impact on the network traffic produced by the query. Consider the query and predicate groups shown in Figure 5. As is described in section 3.4.1, non-local joins are unfolded, so the join predicate is listed in both groups. If group 2 were issued first, all vehicles worldwide within 60 minutes of their destination would respond, probably generating a significant amount of traffic and mostly unnecessary. If group 1 were issued first, only stations in California would respond (indeed, only stations in California will receive the query); using the `S.Names` which are retrieved from this first message, the join predicate can be used as a filter for the second message, and a much smaller number of vehicles will respond.

In order for the query processor to choose a preferred ordering for the messages, and also to choose an appropriate routing key (as described in the section 3.3), we extend the schema definition specification to include a "rank" for each table attribute. The database designer chooses ranks for each attribute based on the semantics of schema, based on the following guidelines:

1. Attributes which partition the records along network topology get highest rank;

2. attributes which have a large number of different values are ranked higher than

3. attributes which have a small number of different values.

There is some art in choosing the best rankings, but it is reasonably straight forward. In our example schema, attributes `Station.State` and `Vehicle.Dest` would have high ranks, and attributes `Package.SpecHandling` and `Vehicle.Status` would have low ranks.

To order the predicate groups within a clause, the non-join predicate with the highest rank is selected, and its group is chosen to be issued first. Joins are then unfolded to determine the order of the rest of the groups. In the case that two predicate groups in the same clause are unjoined, which in SQL indicates a cross-product, the groups can be resolved in parallel. An `IN` predicate (indicating a subquery) is treated as a non-join predicate for purposes of ranking.

3.4.3 Processing the Query

Predicate groups are converted to messages and issued in the order determined by the query decomposer. The presence of sub-queries, OR clauses, and/or cross-products allow for a significant amount of parallelism. As described in section 3.4.1, non-local joins are unfolded and intermediate results are issued along with later messages to filter the number of replies. A predicate group containing an IN predicate waits for the sub-query to be resolved before being issued.

In the case that the first part of a join, or one part of a cross-product, generates no results, the other messages in the clause are not sent; there is no result from that clause. In the case that a sub-query returns no result, the clause containing the `IN` predicate is not resolved; there is no result from that clause.

4. IMPLEMENTATION

Figure 7 outlines the implementation of our Netabase demonstrator system. The purpose of the demonstrator was to prototype our query decomposer and characteristic routing protocol. The data sources were implemented to conform to the characteristic routing and Netabase protocols, but not much effort was made to make them thin in this version.

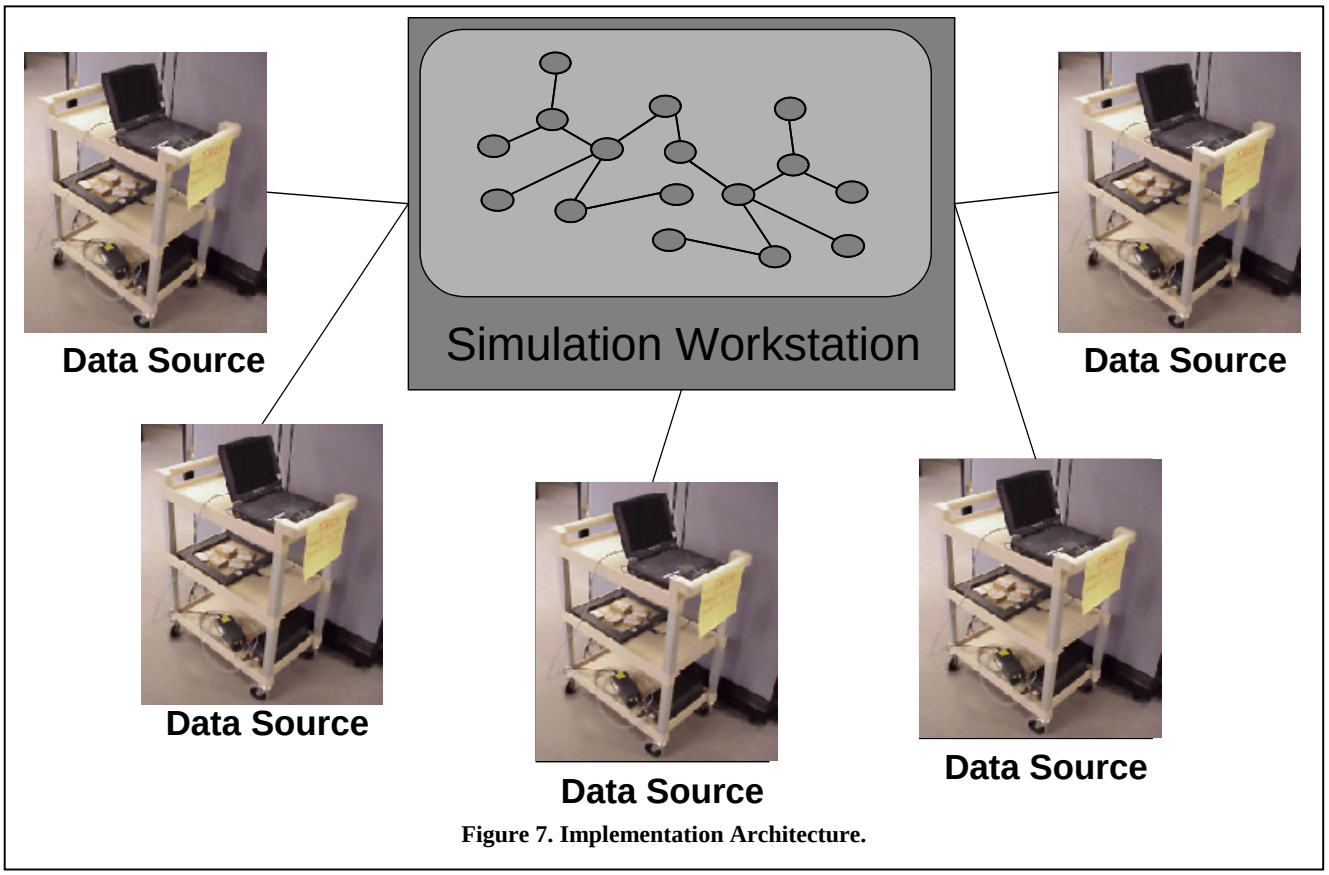

Figure 7. Implementation Architecture.

The data sources in our system were laptops running Microsoft Windows connected to Siemens Moby RF-ID tag readers. We made a collection of small packages, each with an RF-ID tag encoding a `Package` record and the `Customer` records for the sender and receiver. Laptops served the role of vehicles or stations, and the vehicle or station parameters were set through a graphical user interface.

Since scalability was an important concern for us, we simulated a wide area network connecting our data sources. We used the NS network simulator from the University of California at Berkeley [7], which has the capability to simulate part of a network in real-time, while connecting to nodes on live network. Each of our data source laptops was networked to the simulator computer, and traffic between them apparently traveled over a complex network before arriving at its destination. Our characteristic routing software was implemented within the simulator.

Each laptop continuously polled the RF-ID tag reader to keep track of the local inventory. When changes occurred in the inventory, the laptop notified its (simulated) local router of new characteristics, or of characteristics no longer valid. We could move packages from node to node to demonstrate deliveries and pickups.

Several nodes were outfitted with a query module. The query module was implemented as an ActiveX component, taking SQL strings or files as input, and handling all network and query evaluation tasks needed to answer the query. This is an important point for us, because it means that our application only had to understand SQL, and our wide area data management system appeared to the application exactly as a local database would. One application allowed hand entry of SQL queries through a web interface. A second application acted as an airport dispatch optimizer, selecting the best mix of planes and trucks depending on live status of packages discovered through SQL queries on the system.

We tested have our implementation with as many as seven real data sources, and up to 100 simulated data sources, and it performs well. This work is ongoing, and we have good preliminary results with 10^3 data sources.

5. FUTURE WORK AND CONCLUSIONS

We are currently in the process of building a larger simulation to test scales of 10^5 nodes or more. Due to the computational overhead of such a simulation, we are required to distribute the simulation over several computers and develop some new tools for evaluating the results.

We are also in the process of building a deployable prototype of our software, for use by our customer, a large postal logistics solution provider.

Significant work can still be done to improve the efficiency of the system, reduce its footprint. We have, however, implemented a system that has the following interesting properties:

1. Data is entirely distributed to its sources. No central database is used. There is no service bottleneck or central failure point.

2. Anyone connecting to the network (with the proper authentication) can issue an SQL query and receive the results.

3. The basic architecture scales in a manner similar to IP networks.

We believe this method of wide area data management may be quite valuable for sensor networks and other wide area distributed systems in the future.

6. ACKNOWLEDGEMENTS

The authors have benefited from many discussions in the development of Netabase, and we particularly thank Karl-Heinz Maier, Arding Hsu, and Georg Diller for their contributions. Bich Nguyen was instrumental in the development of our first prototype, and Stefan Bindel and Matthias Heiler have made significant contributions in more recent versions.

7. REFERENCES

[1] Bonnet, P., J.Gehrke, and P.Seshadri. "Towards Sensor Database Systems". 2nd International Conference on Mobile Data Management. Hong Kong, January 2001.

[2] Deering, S., "Multicast Routing in a Datagram Internetwork", Stanford Technical Report, STAN-CS-92-1415, Department of Computer Science, Stanford University, December 1991.

[3] Estrin, D., R. Govindan, J. Heidemann and S. Kumar. "Next Century Challenges: Scalable Coordination in Sensor Networks" , ACM MobiCom 99, August 99, Seattle, USA.

[4] Hirsh H., C. Basu, and B. Davison. "Learning to Personalize". Communications of the ACM, August 2000, Vol. 43, No. 8, pp. 102-106.

[5] Kahn, J., R. Katz and K. Pister, "Mobile Networking for Smart Dust", ACM/IEEE Intl. Conf. on Mobile Computing and Networking, ACM MobiCom 99, Seattle, WA, August 17-19, 1999.

[6] Navas, J. and T. Imielinski. "Geographic Addressing and Routing". ACM MobiCom'97, Budapest, Hungary. September 26-30 1997.

[7] "The Network Simulator - ns2." http://www.isi.edu/nsnam/ns.

[8] Ozsu, M. and Valuriez, P. "Principles of Distributed Database Systems", Prentice Hall, 1999.

[9] Radoslavov, P., D. Estrin, and R. Govindan. "Exploiting the Bandwidth-Memory Tradeoff in Multicast State Aggregation". Technical report 99-697. Computer Science Department, USC. July 1999.