Low Cost Databases for NOW

Gianni Conte[1], Michele Mazzeo[2], Agostino Poggi[1],
Pietro Rossi[2], and Michele Vignali[1]

[1] Dipartimento di Ingegneria dell'Informazione
University of Parma, Viale delle Scienze, 43100 Parma, Italy
{conte, poggi, vignali}@ce.unipr.it ,
WWW home page: http://www.ce.unipr.it
[2] ENEA HPCN Project
Via Martiri di Monte Sole, 4, Bologna, Italy
{rossi, mazzeo}@pchpcn1.arcoveggio.enea.it ,
WWW home page: http://www.glue.bologna.enea.it

Abstract. In this paper, we present a system called DONOW (Database on Network of Workstation), that is an example of an implementation and maintenance of a low cost distributed database without performances penalties. The system has a three tier architecture based on a client, a service and a data layer. The client layer allows local and remote users to interact with the system through a Java user friendly interface. The service layer is implemented in C++; it allows the service of user requests and, in particular, the management of queries involving more than one DONOW node. The data layer is based on a well-known freeware relational database management system, that is, MySQL for each DONOW node. DONOW is under experimentation and will be used by Partena, an industry producing very advanced machinery for pharmaceutical products packaging, for the management of the information about their machines, that consists of close to 100000 drawings and related documentation.

1 Introduction

Recent network and computer evolution allows us to obtain low cost high performance by spreading the computation in parallel over a so called Network of Workstations (NOW) [2, 7, 8].

Some NOW architectures have already been presented. However, there is a lack of real application that take advantage of them, mostly because there are not dedicated programming, developing and management tools for NOW architecture.

Databases are one of the most important software products that are used in real applications and that can take advantage of parallelization. Therefore, the availability of database management systems for NOW architectures could be the key for the success of NOW architectures. A database management system for such architectures is a distributed database [16] whose main features

should be to take advantage of recent network evolutions to avoid the bottleneck of communication between nodes and to scale out the system partitioning the databases into clusters with different number of nodes.

This paper is a preliminary report of a project, called DIStributed COmputing on network of PC for data base applications (DISCO), whose purpose is the realization of a NOW database management system called Database On NOW (DONOW). The paper, after a brief presentation of related work, describes the software architecture of DONOW and the hardware supporting it and presents the use of the database for the management of the information about machines for pharmaceutical products packaging.

2 Related work

Even if NOW architectures are quite similar to *shared-nothing* distributed architectures for databases [6, 10, 12, 13], the NOW research community is more interested in the analysis of compute-intensive applications than in the analysis of data-intensive applications and the few works centred on data-intensive applications use parallel sort algorithm to benchmark their prototypes [3, 4].

The only interesting work involving the realization of a database management system on a NOW is under development by the Microsoft Scaleable Servers Research Group is realizing a SQL Server for clusters that will scales up to giant hundred gigabyte databases [15]. However, the result is not yet achieved, moreover, it does not offer two fundamental features of distributed databases, that is, transparent partitioning of data and parallel query decomposition.

3 Hardware architecture

The hardware architecture of DONOW currently comprises four nodes and a front-end; in Fig. 1 there is a schematic representation of it. Each node is an Intel Redwood motherboard with dual PentiumII 300Mhz, 128MByte of RAM a Adaptec AIC-7880 Ultra SCSI host adapter and two network interface cards. The SCSI adapter serves a 4.5GByte hard disk from Quantum. Of the two interface cards, one is an Intel EEpro 100 integrated on board, the other is a 3Com905. The EEpro 100 is connected to a 10Base ethernet HUB and through that to the LAN while the 3C905 connects the nodes through a 100BaseTX SWITCH. The ethernet network is primarily used for system administrative tasks and NFS mounting of user directories, while the fast ethernet network, accessible only within the nodes, is used by the high performance applications. The front end, a pentium 200MMX decouples the nodes from internet, acting as a firewall, while the nodes are locally accessible directly from the LAN connected to the HUB and from some computers connected to the SWITCH.

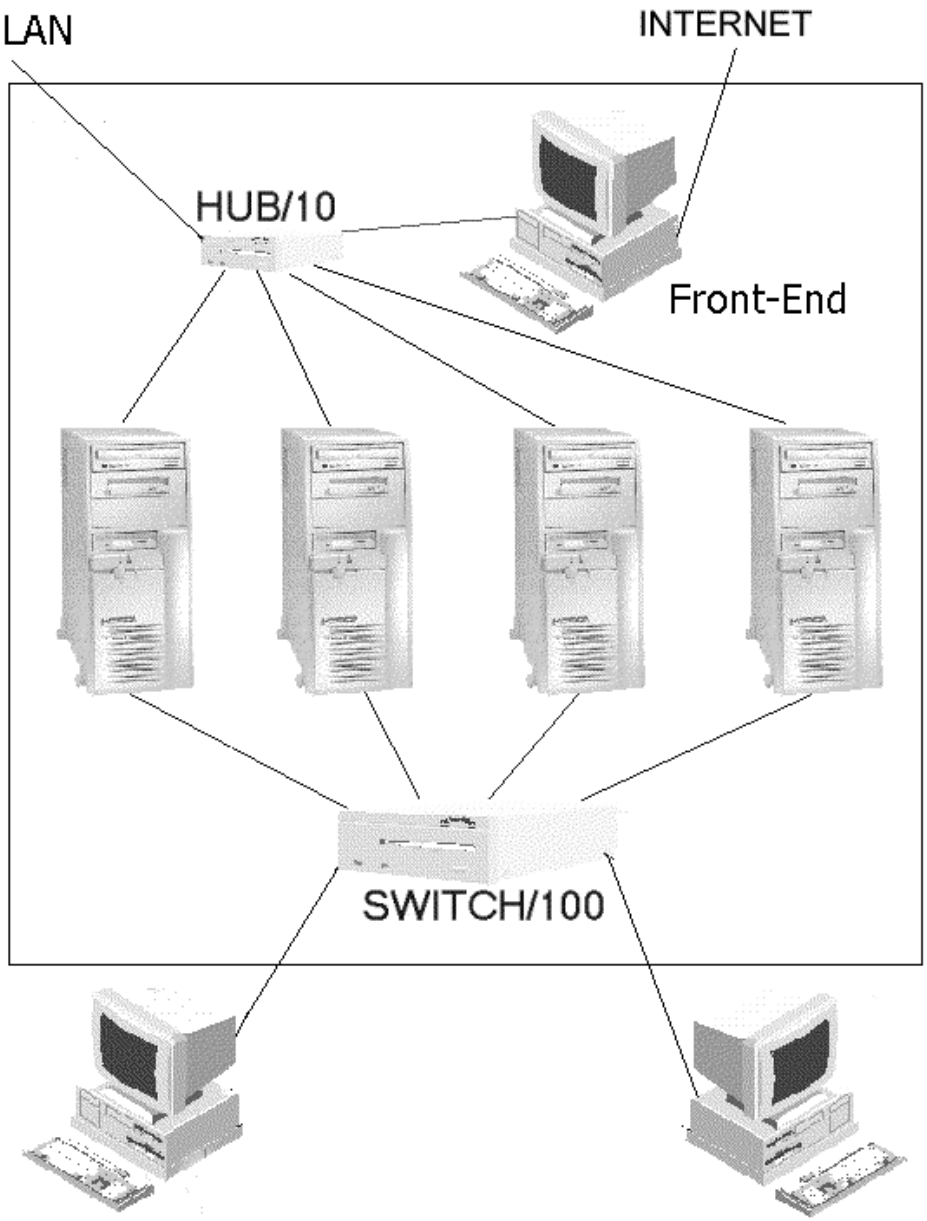

Fig. 1. DONOW cluster architecture

4 Software architecture

As we said before, our goal was the realization of a distributed database for NOW clusters that fits the previous hardware architecture, but also scales for greater clusters.

To guarantee an optimal level of flexibility, usability, scalability, performance and low implementation and maintenance costs, we decided:

1. to realize a multi-layer (three-tier) architecture, distinguishing the client, the service and the data layers;
2. to use a solid pre-existent relational database management system to manage the data in each node of DONOW;
3. to use object-oriented languages for the realization of the software;
4. to use a mix of C++ language, for the critical code of the service and data layers that mainly constraints the performance of the system, and of Java language for the client layer to make accessible the system not only to local clients, but also to remote clients through internet; and
5. to use well-known freeware operating system and software, that is, Linux operating system, gcc++ compiler, jdk 1.1 and MySQL database management system.

4.1 The Client layer

The client layer is based on a set of local clients connected directly to the service layer and a set of remote clients connected to the service layer through an external front-end server.

The front-end server has been introduced because of security and performance reasons. Remote users must be identified and, eventually, refused before accessing the system; moreover, they can interact with DONOW only through some clients implemented as Java applets accessible via an internet browser and so requires the availability of one or more WWW servers. These duties could be managed by each DONOW nodes, but the introduction of an external front-end lightens the DONOW nodes of these duties placing a WWW server on the front-end that manages user authentication and remote clients interaction.

Fig. 2. Two views of DONOW client interface

Clients are Java applications or applets that allow users to interact DONOW database through visual queries without any knowledge about SQL and about the way data are structured in the database. For example, a user of the Partena database can navigate the three of documents related to the design of a machine by simply clicking on the part to expand. Fig. 2 shows a view of the DONOW client interface of the Partena database.

Each client allows a user to interact with the service layer to query and modify stored data; the interaction happens through the exchange of packets, called DoNowPackets.

A DoNowPacket is a serialization of an instance of a class derived from the DoNowPacket abstract class (in its Java meaning). This class has been implemented both in Java and C++, and is able to send/receive itself and guarantees us to have a very flexible asynchronous protocol to exchange control information. Fig. 3 shows the structure of the DoNowPacket abstract class.

Fig. 3. DoNowPacket class diagram

Each new packet must inherit from DoNowPacket and implement send(), receive() and action() methods. For example, if we have to send an SQL query, it is easy to build a packet, called SimpleQuery, like the one in Fig. 4.

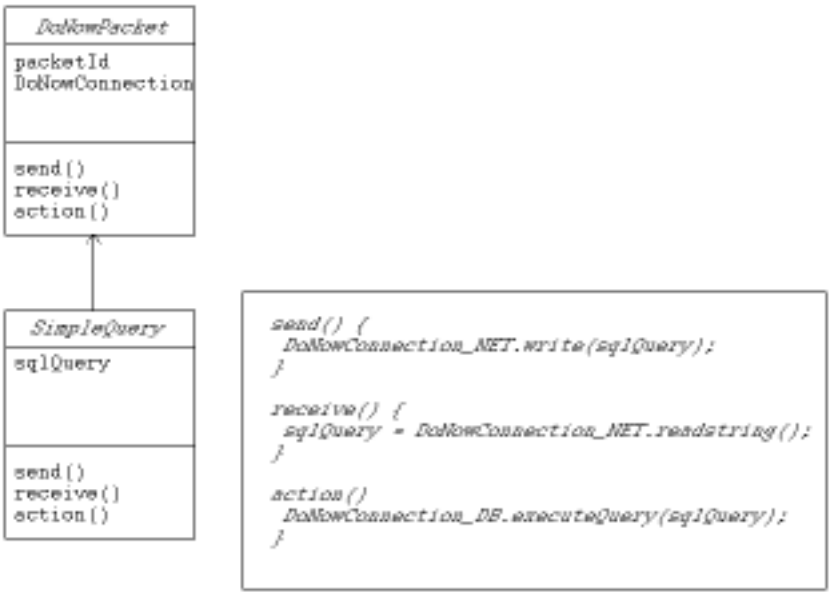

Fig. 4. Example of a simple packet implementation

Each client is connected to the service layer though the internet stream socket protocol. The client does not use primitives of a standard socket libraries, but

instances of a class, called DoNowConnection_NET, whose interface masks the use of a specific communication protocol. This implementation simplifies the possible future use of different communication protocols.

4.2 The Service layer

The service layer is composed of a service demon for each DONOW node. Each demon has the main task to handle the action of DoNowPackets. When a DoNowPacket arrives an action must be taken according to the kind of packet (typically this is a query to be executed or a control message to be sent) and can involve one or more nodes of the data layer.

Clients do not know the number of nodes and their addresses. Clients only know the address of the first DONOW node that is entrusted with balancing the load of the different demons. When it receives a connection request from a client, it returns to the client the address and the listening port of the node with minor load.

Each demon can receive packets from different clients at the same time, therefore, they must be concurrently managed. Due to the importance of performance we choose to handle concurrency by thread. A multi-thread system is more efficient than a multi-process one, because the cost of a context switch is minimal in a threaded architecture. Threads are provided (under Linux) in the form of dynamic linked library. The library used are based on pthread (POSIX thread) version 0.6.

4.3 The data layer

As we said before, the data layer is based on a database management system, that is MySQL [5], for each node of DONOW.

The problem of building a distributed database from scratch is too expensive for our limited human resources and would drive us to a custom solution which will be in few months "out of order software" (OOOS). The choice of a well tested and supported (and documented) database give us a solution with less maintenance problem.

Our solution will be not MySQL dependent, but could be ported to work also with another database. All that the new database must have is a communication layer and a C API; these two characteristics are too important for a database to be not implemented.

MySQL is a multi-thread and sockets based software. That is, it uses threads (and not processes) to handle concurrency and sockets to handle communication between a generic client and the database server.

Because MySQL provides only a C API and we developed critical software layer in C++, we have built a C++ wrapper to its API. This has two main advantages: i) we do not mix C++ with C; and ii) the data layer becomes API independent; in fact, changing API will change only the C++ wrapper and this will have no influence for the rest of the system.

MySQL databases are connected to the service layer through the MySQL protocol. This derives from the fact that is too expensive to rewrite a new high performance MySQL protocol in the first stage of the project. We planned to have a functioning and well modularized prototype of the system at the beginning and then to optimize each component and each protocol. Therefore, at the same way of the interaction between the client and service layers, communication is not based on the use of primitives of MySQL protocol, but on instances of a class, called DoNowConnection_DB, whose interface masks the use of a specific communication protocol and so simplifies the possible future use of different communication protocols.

4.4 Query parallization

A main problem of DONOW system is the management of queries involving data on different nodes. There are two different cases: i) the query can be decomposed in a set of independent subqueries each of them involving data of a single node; and ii) the query cannot be decomposed in a set of independent subqueries each of them involving data of a single node.

In the first case, the execution of the query is simple (Fig. 5 shows a graphical representation of an example of independent subqueries):

1. a client layer node sends a query to a service layer node;
2. the service layer node decomposes the query and sends the subqueries to the data layer nodes involved;
3. the data layer nodes replay the results of the subqueries;
4. the service layer node collects the results and sends the results to the client layer node.

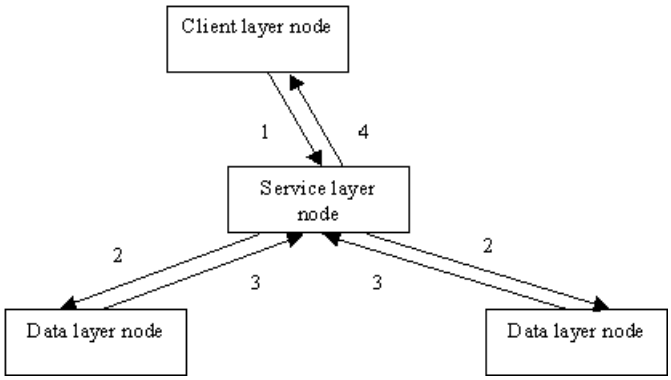

Fig. 5. An example of indenpendent subqueries

In the second case, the execution of the query is quite complex (Fig. 6 shows a graphical representation of an example of non-independent subqueries):

1. a client layer node sends a query to a service layer node;

2. the service layer node decomposes the query into a tree of subqueries: leaf subqueries involve data of a single node, the other subqueries (composition subqueries) work on the results obtained by the queries of their subtree;
3. the service layer node sends the leaf subqueries to the data layer nodes involved;
4. the data layer nodes replay with the size of their results;
5. for each composition subquery:
   (a) the service layer node chooses the data layer node declaring the greatest results size;
   (b) the service layer node moves the other results to the chosen data layer node;
   (c) the service layer node sends the composition subquery to the chosen data layer node;
   (d) the chosen data layer node replay with the size of its results;
6. after the execution of the root composition subquery, the service layer node gets the results of the query and sends them to the client layer node.

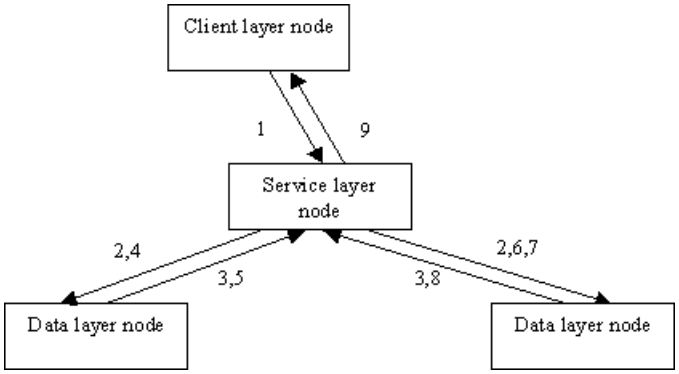

Fig. 6. An example of non-indenpendent subqueries: a join is performed in the data-layer node in the right of figure

5 A case study

The solution proposed arose out of the attempt to satisfy the need of a typical mechanical company, Partena, to manage the information about the design of its products.

Partena produces very advanced machines for pharmaceutical packaging. Such machines are made up of a subset of common parts, more or less standardized, and of a very large number of parts dealing with the specific nature of the needs of their customers. The design of a machine is a top-down process realizing a *tree* of documents each of them corresponding to the design of one of its parts. Parts are of five different types: P02, P03, P13, P40 and P50. P02 are composed of P03, P13, P40 and P50 parts. P03 and p13 are composed of P40 and P50 parts. Finally, P40 and P50 are leaf elements. Therefore, Partena must maintaining an ever growing database given by a graph of documents (some parts and their documentation are reused for realizing

different machines) composed of technical designs, specs and ad hoc solutions for the different machines that Partena built. Fig. 7 presents a graphical representation of a part of Partena database.

This, rather quickly growing database, has to be consulted every time a maintenance operation needs to be performed or a new project needs to be undertaken because of the use of common parts in different machines. Presently, the database consists of close to 100000 drawings and is fully kept on paper. Tracking down the complete documentation relative to one single machine is an operation keeping one person busy for about an hour and producing always from 10 to 15 drawings plus several inventory listings, and a handful of technical documents detailing the reasoning behind the adopted solutions.

The database, built so far, is vital to the functioning of the company and is bound to be the bottleneck of their operation if some procedures that scales better with complexity are not adopted. The obvious solution, proposed with this application of DONOW technology, was to develop a way to digitize their database, and provide some form of digital storage and retrieval.

The design goals of such system has taken into account that the database will keep growing at a fast pace, some data mining activity should be made possible to exploit more effectively the historical knowledge associated with the solution of projects realized in the past and the possibility to include, in the near future, several other application with minimal software tunings.

Besides it, other three design goals are: i) to query the database via simple internet connections, ii) to enable remote service units to intervene without asking someone at the headquarters to search materials for them and iii) to produce a cost effective solution both as initial investment and in fixed maintenance costs.

Given the graph structure of Partena database, an object representation and an object-oriented database management system seem be the most suitable solution for their management. This solution is easily feasible if we would manage them on a centralized database management system. However, it is realized with difficulty on a distributed database because of the need of techniques, that are not yet well studied, to partition the graph on different nodes of the database management system guaranteeing a good performance of the system.

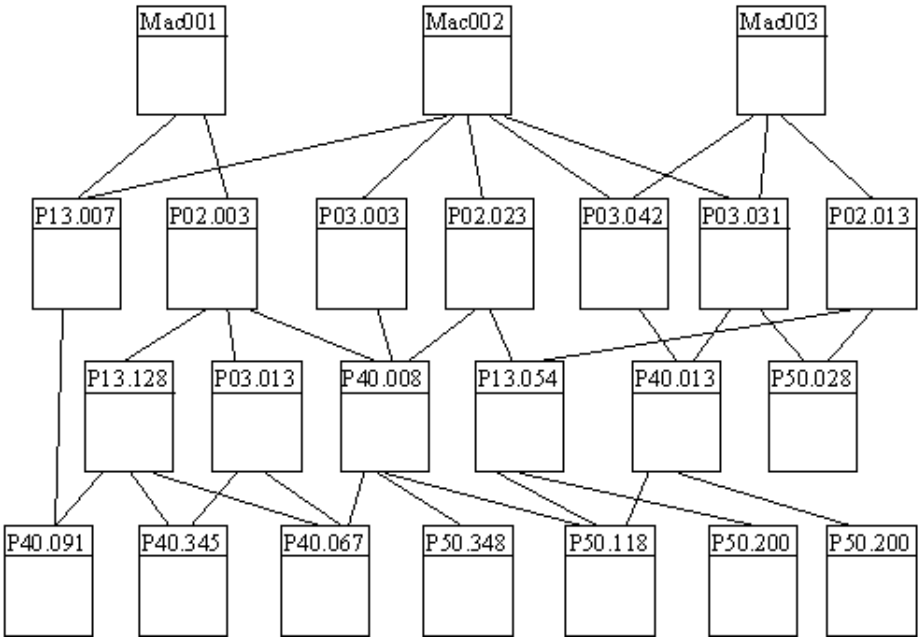

Fig. 7. graphical representation of some Partena machines documents

A relational database is not the most appropriate for graph structures. However, such structures can be managed through the use of join tables [1] obtaining a set of tables that can be easily partitioned for a distributed database management system (see Fig. 8).

We use horizontal partitioning [9] for all the tables, that is, all the nodes manage all the types of table, but each node manages a different subset of records (see Fig. 9). This solution simplifies a lot DONOW software because it does not realized concurrency control and commit protocols to manage distributed transactions. Such protocols are necessary when an operation must modify the data of different nodes of the database management system. Deletion and modification are few used operations and usually acting on a single record, that is, managed by a single node, that is, by a MySQL system. Moreover, the insertion of new data can be decomposed in a sequence of record insertions each of them managed by a MySQL system. However, data insertion is transparent to the user: the DONOW service layer decided on which node inserts the different records.

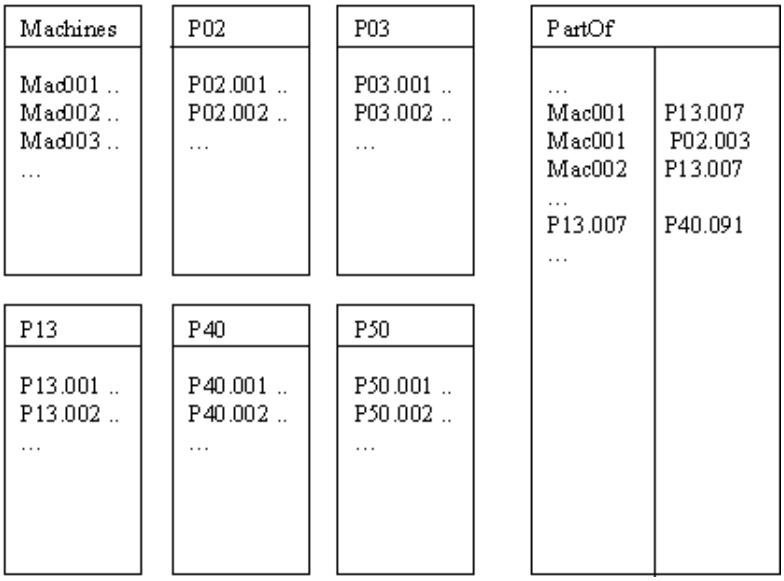

Fig. 8. Relational representation of some Partena machines documents

Such a partitioning simplifies a lot query parallelization too. In fact, the duty of a DoNowPacket action is always to send the same query to all the data layer nodes and collects the results.

Currently, Partena is digitizing the design machine documents and we are experimenting the system on a limited set of data. The full data will be available at the end of the year; therefore, complete experimental results will be available at the beginning of the 1999.

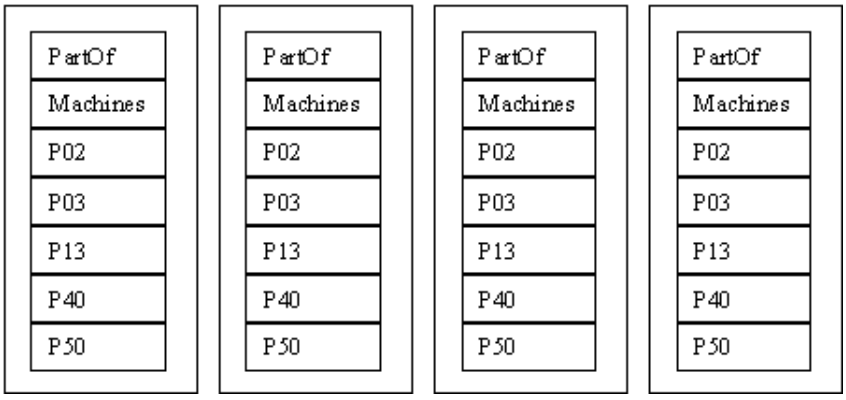

Fig. 9. Partitioning of tables on the DONOW system

6 Conclusions

This paper, presented a system called DONOW (Database on Network of Workstation), that is an example of an implementation and maintenance of a low cost distributed database without performances penalties.

DONOW has a three tier architecture based on a client, a service and a data layer. The client layer allows local and remote users to interact with the system through a Java user friendly interface. The service layer is implemented in C++; it allows the service of user requests and, in particular, the management of queries involving more than one DONOW node. The data layer is based on a well-known freeware relational database management system, that is, MySQL for each DONOW node.

The three layer architecture, the object-oriented implementation through C++ and Java languages, and the use a conventional relational database for each node of the cluster introduces a significant overhead that could in principle reflect itself on a poor performance of the system. For example, the use of a two layer architecture, that eliminates the service layer, where clients would interact directly with MySQL nodes through their JDBC interface, could offer better performance, and certainly would, on modest sized data sets.

However, our choices allowed us to achieve in short time a very important design goal, that is, to have a low cost and flexible prototype that would allow the user to start deploying the new tool, and us to experiment different solutions for the realization of a database management system for NOW architectures.

The exploitation of a relational database management system, that is, MySQL, for each DONOW node, goes a long way towards achieving a short delivery time of a prototype.

The three layer architecture and the use of object-oriented technologies allows us to experiment different solutions reusing most of the software and quickly prototyping different communication protocols and parallelization algorithms without intervening on the user interface. For example, replacing the TCP/IP communication protocol with another protocol causes only the redefinition of the send() and receive() methods.

Finally, the use of Java for the client interface, provides an easy access to the system via internet implementing the client as applet accessible via a WWW server.

Planned future work consists of experimenting different communication protocols between service layers and data partitioning strategies.

As far as protocols are concerned, we are planning to test the performance of PRP [14] and Gamma protocols [11]. Within our framework, that can be simply achieved by specializing the send and receive methods of the DoNowPacket class.

With regard to data partitioning, we are planning to experiment vertical partitioning and some different types of hybrid partitioning not only to increase performance, but also to offer replication of data.

Acknowledgements

We wish to thank the anonymous referees for their valuable comments.

The work has been partially supported by a grant of the European Union (Tender to III/97/31 Lot 5 - Cluster Computing for data Intensive Application) in the framework of the DISCO (DIStributed COmputing on network of PC for data base applications) project and by a grant of the Italian *Ministero dell'Università e della Ricerca Scientifica e Tecnologica* (*MURST*) in the framework of the MOSAICO (Design Methodologies and Tools of High Performance Systems for Distributed Applications) project.

References

1. Ambler, S.W.: Tips for Mapping Objects to Relational Databases. AmbySoft Inc. (1998) Available from http://www.AmbySoft.com/onlineWritings.html
2. Anderson, T.E., Culler, D.E., Patterson, D.A.: A case for networks of workstations: NOW. *IEEE Micro*, 15(1) (1995) 56–64
3. Arpaci-Dusseau, A.C., Arpaci-Dussuau, R.H., Culler, D.E., Hellerstein, J.M., Patterson D.A.: High-Performance Sorting on Networks of Workstations. In *Proc. SIGMOD '97*. Tucson, AZ (1997) 243–254
4. Arpaci-Dusseau, A.C., Arpaci-Dussuau, R.H., Culler, D.E., Hellerstein, J.M., Patterson D.A.: Searching for the Sorting Record: Experiences in Tuning NOW-Sort. In 2nd SIGMETRICS Symposium on Parallel and Distributed Tools. Welches, OR (1998)
5. Axmark, D.: *MySQL 3.22 Reference Manual.* (1998) Available from http://www.mysql.com
6. Baru, C., Fecteau, G., Goyal, A., Hsiao, H., Jhnigran, A., Padmanabhan, S., Wilson, W.: An Overview of DB2 Parallel Edition. In *Proc. SIGMOD '95*. San Jose, CA (1995) 460-462
7. Basu, A., Buch, V., Vogels, W., von Eicken, T.: A User-Level Network Interface for Parallel and Distributed Computing. In *Proc. 15th ACM Symp. on Operating System Principles* Copper Mountain, CO (1995)
8. Blumrich, M.A., Li, K., Alpert, R., Dubnicki, C., Felten, E.W.: Virtual Memory Mapped Network Interface for the SHRIMP Multicomputer. In *Proc. of the Int. Symp. on Computer Architectures* Chicago, IL (1994) 142-153
9. Bobak, A.R.: *Distributed and Multi-Database Systems.* Artech House (1995)
10. Boral, H., Alexander, W., Clay, L., Copeland, G., et al: Prototyping Bubba, a Highly Parallel database system. *IEEE Trans. on Knowledge Data Engineering.* 2(1) (1990) 44–62
11. Chiola, G., Ciaccio, G.: Implementing a low cost, low latency parallel platform. *Parallel Computing.* 22 (1997) 1703–1717
12. DeWitt, D., Ghandeharizadeh, S., Schneider, D., Bricker, A., et al: The Gamma Database Machine Project. *IEEE Trans. on Knowledge Data Engineering*, 2(1) (1990) 4–24
13. Gerber, R.: Informix Online XPS. In *Proc. SIGMOD '95*. San Jose, CA (1995) 463
14. Marenzoni, P., Rimassa, G., Vignali, M., Bertozzi, M., Conte, G., Rossi, P.: An operating system support to low-overhead communications in NOW clusters. In *Proc. of Communication and Architectural Support for Network-Based Parallel Computing*, San Antonio, TX (1995) 130–143

15. Microsoft: *Clustering Support for Microsoft SQL Server High Availability for Tomorrow's Mission Critical Applications.* (1997) Available from http://research.microsoft.com/ gray/.
16. Ozsu M.T., Valduriez, P.: *Principles of Distributed Database Systems.* Prentice Hall, Englewood Cliff (1991)