Annals of DAAAM for 2011 & Proceedings of the 22nd International DAAAM Symposium, Volume 22, No. 1, ISSN 1726-9679
ISBN 978-3-901509-83-4, Editor B. Katalinic, Published by DAAAM International, Vienna, Austria, EU, 2011
Make Harmony between Technology and Nature, and Your Mind will Fly Free as a Bird
Annals & Proceedings of DAAAM International 2011

AN ARCHITECTURE FOR DISTRIBUTED DATABASES ON WORKSTATIONS

BOICEA, A[lexandru]; MAGDALINA, C[atalin]; IONESCU, D[iana] - C[ristina]; RADULESCU, F[lorin] & POPA, G[eorge] D[an]

Abstract: *Many recent research projects and reports on distributed database systems (DDBS) deal with performance. Organizational aspects, such as the distribution of control, or the management of data in an enterprise, rarely influence the design of these DDBS. Furthermore, architectural proposals for the use of the special properties of workstations, which are largely independent but connected by a local area network are lacking. In this thesis, a simple and flexible system architecture is proposed. This proposal is tailored to the workstation environment and it includes both organizational and technical aspects of data distribution, replication and integrity.*
Key words: *distributed databases, architecure, workstation, performance*

1. INTRODUCTION

The goal of this thesis is the development of a simple, but powerful architecture for database services in a LAN environment with workstations. The architecture emphasizes organizational aspects in addition to the technical discussion (Darabant, 2009).

An attractive working environment for workstations includes different services offered via LAN. Besides laser printers, mass storage media and electronic mail, collections of structured data, i.e. databases, should also be available to the user community. For such a database service to be practically employed, it must be attractive to users. This entails providing access to additional information from others without much impeding operations on own data which are made accessible to others.

2. SERVER ARCHITECTURE

Three main components can be distinguished in figure 1: SERDS is the DBMS server, SNC implements database-specific protocols on top of the general LAN software on Lilith, called CIPON, and SAM is the main program coordinating all the activities on the server. A short summary will be given as follows.

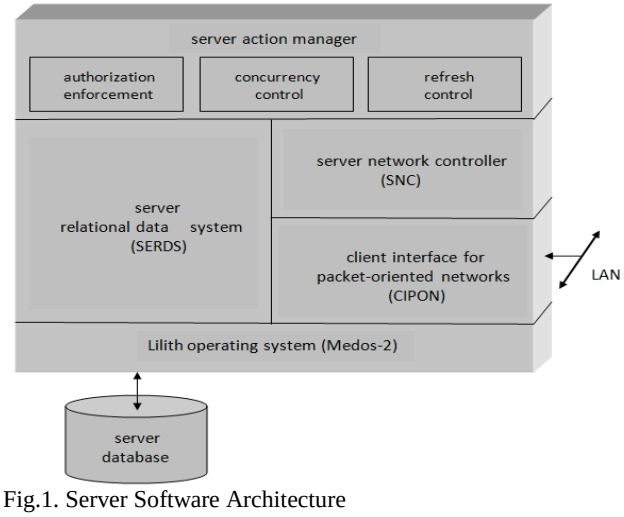

Fig.1. Server Software Architecture

SERDS (server relational data system) is a multi user DBMS which was developed starting from RDS. Similarly to RDS, SERDS offers an interface which allows navigational operations on single tuples of a relation. The fact that SERDS is used as a backend system only somewhat simplified its architecture and interface. The main change regarding RDS was the removal of the implicit dependency from the single user situation, in order to allow SAM to control several concurrent transactions. This included delegating the maintenance of current positions in relations to the clients and implementing an operation-oriented log instead of page-level logging.

SNC (server network controller) implements a protocol which allows the server to receive requests and transmit answers. SNC is based on CIPON which offers a connection-oriented protocol to send and receive fixed length messages.

SAM (server action manager) is the main program on the server. As such, its main task consists of scheduling the requests received from SNC for execution by SERDS and of passing back the corresponding answers. Three special tasks are included in SAM but implemented in separate modules:

- authorization enforcement uses the design of user authorizations as recorded in the meta database to check whether incoming requests are legal;
- refresh control handles requests to refresh duplicates residing on the server and delivers status information to clients, allowing them to decide on the necessity and urgency of a refresh;
- concurrency control.

3. CLIENT ARCHITECTURE

The software architecture is shown in figure 2.

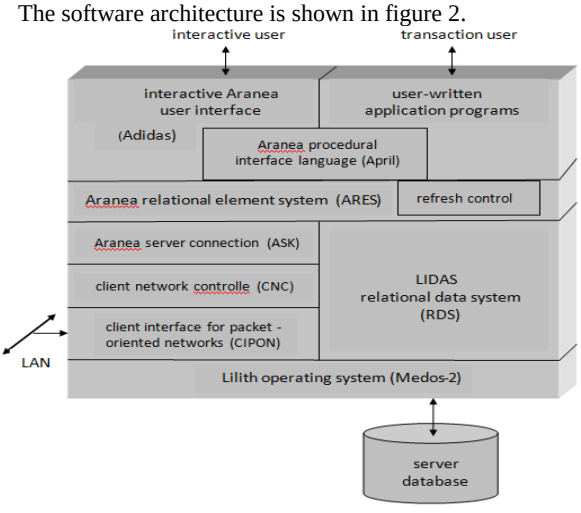

Fig.2. Client Software Architecture

The server interface component was split into two parts: CNC (client network controller) implements the protocols needed for the exchange of requests and answers, but does not care about the content of the messages transmitted and

received. It is the counterpart of SNC on the server side. ASK (Aranea server konnection) translates database-oriented procedure calls to messages that can be transmitted by CNC and unwraps the responses. It thus constitutes the counterpart of SAM on the server side. The interface offered by ASK must be functionally identical to the one of the client DBMS. To this end, ASK has to fulfill some functions which were delegated from the server, such as maintaining current positions in relations.

The decomposition and integration component is implemented by ARES. Its main task consists of mapping the different data descriptions and procedures of ASK and RDS onto a uniform interface. It was decided that the data currency should be shown in this interface (but not the physical location of the data). Thus, a user can specify whether he wants to access an original or a duplicate, but he does not need to care where the data are physically allocated (on the server or locally). To support autonomous operation of user workstations, the default is always the local copy of some data (if one exists at all), be it an original or a duplicate. This allows running as many applications as possible even if the server is not accessible. Another aspect of ARES is refresh handling.

Application programs (implementing predefined transactions to be executed by users who only fill in some parameters) can be written by using the ARES interface only. However, specifying complex queries by using single tuple operations is quite intricate; this is why a relational algebra interface component (April) is implemented. But, at least for updates, a direct access to ARES is still necessary. Finally, Adidas (Aranea distributed database system) is an interactive window-oriented user interface to April and ARES. It includes, in a similar way as on the server, a graphical representation of the software architecture indicating the component currently at work.

4. IMPORTANT CONCEPTS

After this general description of the software architecture, we will now show the implementation of certain important concepts. (Auvray, 2008)

Refresh **-** A refresh always involves a user machine and the server. The refresh instant depends on the presence of users in the system. It is therefore obvious to have the refresh process controlled by the client. This is also consistent with the nature of the server which never becomes active by itself, but rather reacts to requests it receives from clients.

A refresh takes place in two phases: first, server and client have to agree that a refresh is necessary and that the time is right to do it; second, if a refresh has to take place at all, the client starts a special transaction during which it reads or writes the data fragment to be refreshed, replacing the old data. The user is also given the possibility of requesting an immediate refresh explicitly. In this case, the first phase is skipped.

The goal of the first phase is to determine whether a refresh has to take place at this moment. To come to this decision, cooperation between server and client is necessary. It is the client which initiates the refresh and it must also make the decision when to refresh. To this end, it needs two kinds of information: first, the refresh necessity and urgency conditions and second, the current state of server and client. So, the client has to look up some information in the meta database and to pass a message to the server, requesting the necessary information. In the second phase, the update of some data fragment actually takes place, whole fragments being replaced by new data. In case of refreshing from client to server, special attention must be paid to process effects on the server. Therefore, a special command must be given to the server. In the other direction, regular read requests are issued to the server.

Authorization **-** The authorization is enforced on two levels: first, the run time data descriptions which are necessary to access server data are only delivered to authorized users (or, more precisely, to their clients). Thus, at the time when a connection between server and client is established, authentication information is passed from client to server, and in turn, some descriptions of data stored on the server are returned. Because a server may not trust clients (they are running on a user workstation and can thus be corrupted), a second step must always be added: every data access request submitted to the server is filtered by SAM through a component checking whether the request is legitimate or not and blurring those attribute values of retrieved tuples which the user is not allowed to see. The first step merely serves to deny some of the unauthorized access attempts already on the user machine, thus discharging the server.

The meta database had to be extended to cover authorization aspects (just as for refresh parameters). This meta information is frequently used at run time in order to determine user rights for certain operations. Using a database for these lookups slows down server operations significantly. So, resident tables should be used for authorization information. (Boicea & al., 2010)

Concurrency Control **-** The optimistic control method allowing extended use of duplicates within update transactions is not suited for implementation in a DBMS offering navigational access to data, such as SERDS. The reason is the difficulty of consistently describing the read set for operations which are based on some current position in a relation. A set-oriented interface between server and client is necessary to implement this method (Huang et al., 2001).

5. CONCLUSIONS

A functional specialization between server and clients simplifies the system by separating concerns and thus increasing the modularity of the software. It is interesting to see that this seems to be a general trend in distributed database systems: in their attempt to provide an integrated view of the database by replicating functionality over all participating nodes, the SDDS approaches generally stay on a research level.

Most systems which are heavily used, e.g. in banks or airlines, are loose, heterogeneous federations of largely independent systems with well defined, thin interfaces. The reason lies both on the technical, system-oriented and on the managerial, data-oriented level, and is named simplicity on both levels. Functional specialization, together with well defined interfaces and distributed control, can help to build modular systems which can more easily be designed and managed.

The architecture which was described shows that the explicit classification of data with regard to their ownership (private vs. shared) and of different degrees of data currency (originals vs. duplicates) can serve to match existing organizational methods and structures. This helps not only to improve the acceptance of such systems, it also facilitates the management and maintenance of databases.

6. REFERENCES

Auvray, S. (2008). Just Another Distributed Database? Not Really, *Available from:* http://www.infoq.com/news/2008/04/distributed-dbstrokedb *Accessed on:* 2011-03-15

Boicea, A. Badalau, C., Petcu,R., Nicula, A. (2010) *Improving Application Performance Through Database Caching at Application Tier Level*, The 21th International DAAAM Symposium , Croatia

Darabant, A. (2009). *Proiectarea bazelor de date distribuite*, Ed. Casa cartii de Stiinta, ISBN 9789731336057, Cluj Napoca, Romania

Huang, Y & Chen, J. (2001). *Fragment Allocation in Distributed Database Design*, Journal Of Information Science And Engineering 17, pp. 491-506