N91-25957

HETEROGENEOUS DISTRIBUTED DATABASES: A CASE STUDY

Tracy R. Stewart
Ravi Mukkamala
Department of Computer Science
Old Dominion University
Norfolk, Virginia 23529.

1 INTRODUCTION

The purpose of this case study is to review alternatives for accessing distributed heterogeneous databases and propose a recommended solution. Our current study is limited to the Automated Information Systems Center at the Naval Sea Combat Systems Engineering Station at Norfolk, VA. This center maintains two databases located on Digital Equipment Corporation's VAX computers running under the VMS operating system. The first data base, ICMS, resides on a VAX 11/780 and has been implemented using VAX DBMS, a CODASYL based system. The second database, CSA, resides on a VAX 6460 and has been implemented using the ORACLE relational database management system (RDBMS).

Both databases are used for configuration management within the U.S. Navy. Different customer bases are supported by each database. ICMS tracks U.S. Navy ships and major systems (anti- air, weapons launch, etc.) located on the ships. CSA tracks U.S. Navy submarines and the major systems located on the submarines (anti-sub, sonar, etc.). Even though the major systems on ships and submarines have totally different functions, some of the equipment within the major systems are common to both ships and submarines.

2 PROBLEM STATEMENT

Even though the two databases are physically distinct, there a number of external actions which affect the data in both databases. Keeping the data consistent across these databases is a major problem. For example, the same computer is used within many major systems on ships and submarines. Thus, both CSA and ICMS maintain this information about this computer in their respective databases. If the U.S. Navy decides to stop buying parts for this computer from the existing supplier and start buying from an alternate supplier, this information needs to be updated in both databases. It should be noted that the two VAX's communicate via DECNET and that critical data must be updated in real-time up to 2 hours; all other data must be updated on a daily basis.

3 PROPOSED ALTERNATIVES

We propose two alteranatives.

(1) A global data manager (GDM) resides on one machine and all update transactions (to either of the systems) are routed through this central location. Any transaction that affects only one database is sent directly to the affected database. Since non-critical updates may be propagated on a daily basis, it is possible to send them directly to the affected database.

(2) Require each system to log all local transactions. Each system would have a local agent GDM that would regularly review each local database log and determine if a remote database transaction should be generated. The GDM would also report in the database log, the status of any transactions issued remotely. This alternative is the most viable for this environment. The GDM is primarily an initiator of transactions to keep the databases synchronized. The GDM also monitors the results of the initiated transactions and records the results in the originating database log. The local data managers (LDM) would be solely responsible for local transactions and local concurrency control.

The latter alternative appears to be more suitable for the current environment.

4 Sample Data

The ICMS database consists of records, data items and sets. An example of affected records sets and data items is given below.

```
SCHEMA NAME IS ICMS_SCHEMA

RECORD NAME IS SITE_DESCRIPTION
    WITHIN SITE_INFO
    ITEM SITE_IDENTIFICATION TYPE IS CHARACTER 15

RECORD NAME IS SYSTEM_DESCRIPTION
    WITHIN SYSTEM_INFO
    ITEM SYSTEM_NAME TYPE IS CHARACTER 26
    ITEM SYSTEM_AINAC TYPE IS CHARACTER 2
    ITEM SYSTEM_FSCM TYPE IS CHARACTER 5
    ITEM SYSTEM_PART_NO TYPE IS CHARACTER 26
    ITEM SYSTEM_STOCK_NO TYPE IS CHARACTER 26
    ITEM SYSTEM_CATEGORY TYPE IS CHARACTER 5

RECORD NAME IS SYSTEM_INSTALLATION_STATUS
    WITHIN INST_AREA
    ITEM SYSINST_STATUS TYPE IS CHARACTER 1
    ITEM SYSINST_SITE_ID TYPE IS CHARACTER 15
    ITEM SYSINST_SYSTEM_NAME TYPE IS CHARACTER 26
    ITEM SYSINST_SERIAL_NUMBER TYPE IS CHARACTER 15

SET NAME IS ALL_SITES
    OWNER IS SYSTEM
    MEMBER IS SITE_DESCRIPTION
     INSERTION IS AUTOMATIC RETENTION IS FIXED
     ORDER IS SORTED BY
         ASCENDING  SITE_IDENTIFICATION
         DUPLICATES ARE NOT ALLOWED

SET NAME IS SITE_SYSTEM_INST_STAT
    OWNER IS SITE_DESCRIPTION
    MEMBER IS SYSTEM_INSTALLATION_STATUS
     INSERTION IS AUTOMATIC RETENTION IS FIXED
```

```
     ORDER IS FIRST

SET NAME IS SYSTEM_SITE_INST_STAT
    OWNER IS SYSTEM_DESCRIPTION
    MEMBER IS SYSTEM_INSTALLATION_STATUS
     INSERTION IS AUTOMATIC RETENTION IS FIXED
     ORDER IS SORTED BY
         ASCENDING  SYSINST_SERIAL_LETTERS
                    SYSINST_SERIAL_NOS
         DUPLICATES ARE LAST

SET NAME IS ALL_SYSTEMS
    OWNER IS SYSTEM
    MEMBER IS SYSTEM_DESCRIPTION
     INSERTION IS AUTOMATIC RETENTION IS FIXED
     ORDER IS SORTED BY
         ASCENDING  SYSTEM_NAME
         DUPLICATES ARE LAST
```

The CSA database consists of tables and data items. Examples of tables and data items are given below.

```
TABLE: SITE

 Name                            Null?    Type
 ------------------------------- -------- ----
 SITE_IDENTIFIER                          CHAR(15)
 SQUADRON                                 CHAR(30)
 CONFIG_DATA_MANAGER                      CHAR(20)
 TYPE_COMMANDER                           CHAR(14)
 FLEET_CODE                               CHAR(1)
 STD_NAVY_DIST_LIST                       CHAR(8)
 LOCATION                                 CHAR(30)


TABLE: NOMEN_ITEM

 Name                            Null?    Type
 ------------------------------- -------- ----
```

CATEGORY_TYPE_CODE	NOT NULL	CHAR(5)
PROGRAM_MANAGER		CHAR(14)
TECHNICAL_MANAGER		CHAR(14)
SYSTEM_INTEGRATION_AGENT		CHAR(14)
DESIGN_AGENT		CHAR(14)
ACQUISITION_ENG_AGENT		CHAR(14)
TECHNICAL_DIRECTION_AGENT		CHAR(14)
IN_SERVICE_ENG_AGENT_CODE		CHAR(14)
IN_SERVICE_ENG_AGENT_PHONE_NUM		CHAR(12)
IN_SERVICE_ENG_AGENT		CHAR(14)
SPCC_ITEM_MANAGER_PHONE_NUMBER		CHAR(12)
SPCC_ITEM_MANAGER		CHAR(40)
DATE_UNDER_CONFIG_CONTROL		DATE
DATE_OF_LAST_UPDATE		DATE
TEST_AND_EVALUATION		CHAR(14)
BRIEF_NAME		CHAR(50)
FULL_NAME		CHAR(70)
NOMEN	NOT NULL	CHAR(26)

TABLE: SITE_INSTALLATION

Name	Null?	Type
BELONGS_TO_SITE_ID	NOT NULL	CHAR(14)
SERIAL_NBR		CHAR(15)
SERIAL_NUMBER		CHAR(15)
SERVICE_APPLIC_CODE		CHAR(10)
NOMEN		CHAR(26)

5 IMPLEMENTATION

The following implementation scheme is proposed to implement the second method.

A. Log all local transactions that relate to both databases. There will be one log per database and the log contains a timestamp of when the local transaction was committed, the node name, transactions (add, update, delete) and records/tables affected and data items affected. The key to all transactions is the system/unit nomenclature.

B. Modify each application program, that alters data, to log each transaction after it is committed. Sub-transactions must be entered into the log in the order they were executed on the local machine.

C. Generate list of system/equipment nomenclatures that are common between each database. The list would reside in a centralized location that would contain nomenclatures that are shared between each database. The Database Administrator (DBA) would maintain the nomenclature list.

D. Develop GDM to review the log files and search for transactions against common nomenclatures. After the GDM determines the transaction must be issued globally, it generates a remote transaction to be executed on the remote machine. GDM must know how to translate a transaction to be executed on another database.

E. The GDM must formulate the transaction in the appropriate data manipulation language so it can be executed on the affected database.

F. The GDM must send through DECNET a transaction package that will execute on the remote machine as a batch job. Set up account on each system for remote system to use to go into and submit batch jobs from.

G. The LDM will execute the transaction package as if it were generated locally.

H. When the transaction completes, the batch program will send a message back to the originating machine signifying that the transaction has completed. Once the message is received, the transaction is removed from the log. Generate a process to run on the other machine to update the log file.

I. Crash Recovery Procedures - As soon as a system recovers from a crash, a message will be sent to the other system to signal that processing can continue. The GDM will resend any transactions packages that it did not receive a completion on. If database commits and then crashes before sending a message, need to determine what to do.

6 RECOMMENDATIONS

The situation exists that two data bases that have multiple occurrences of the same data must learn to co-exist and keep each other informed of any changes to their duplicate data by keeping a log file, the internal database processing software is not affected. The application programs are responsible for logging the affects of the program.