Copyright © 1982 American Telephone and Telegraph Company
THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 61, No. 7, September 1982
Printed in U.S.A.

Stored Program Controlled Network:

Data Base Administration System—Architecture and Data Base Design

By S. F. SAMPSON and D. W. TIETZ

(Manuscript received July 6, 1981)

The Data Base Administration System (DBAS) maintains a data base of up to 12 million entries, and supports an update rate on these entries of up to 7,000 per hour using magnetic tape, direct data link, or clerk terminal entry. The DBAS software architecture and data base design to support these requirements are described in this paper. The architecture was developed using real-time functional process modeling techniques. Software process concurrency and modular separation of the major software functions optimize throughput and permit quick clerk terminal response.

I. INTRODUCTION

The No. 2 Data Base Administration System (DBAS) administers data bases stored in Automatic Intercept Systems (AISs), Traffic Service Position Systems (TSPSs) and Billing Validation Applications (BVAS). This administration consists of providing an initial load of data, ongoing updates, audits, and various reports. In general, the No. 2 DBAS serves as the telephone company's (Bell operating and, possibly, independent telephone companies) interface to data stored in these aforementioned systems.

1.1 Transition from No. 1 DBAS to No. 2 DBAS

The No. 2 DBAS supersedes the No. 1 DBAS (previously known as the File Access Subsystem, FAS) and subsumes the responsibilities of the No. 1 DBAS for administering data in the AIS. This is accomplished by retaining the application software used on the No. 1 DBAS, and running it on the No. 2 DBAS processor in timeshare with new software that was written to administer TSPS and BVA data bases.

Retaining the original software to perform the AIS administration was less costly than writing a totally new set of programs for this purpose, even though modifications to the existing software were required.

1.2 Basic software architecture

The software architecture for the No. 2 DBAS is shown in Fig. 1. For the purposes of this paper, software architecture will be defined as the division of the application software into processes, plus the design of the interprocess communication. A process, in the *UNIX** operating system sense, is a running instance of a program.[1] Several processes may share one program text, and are distinguishable only by the data on which they are working.

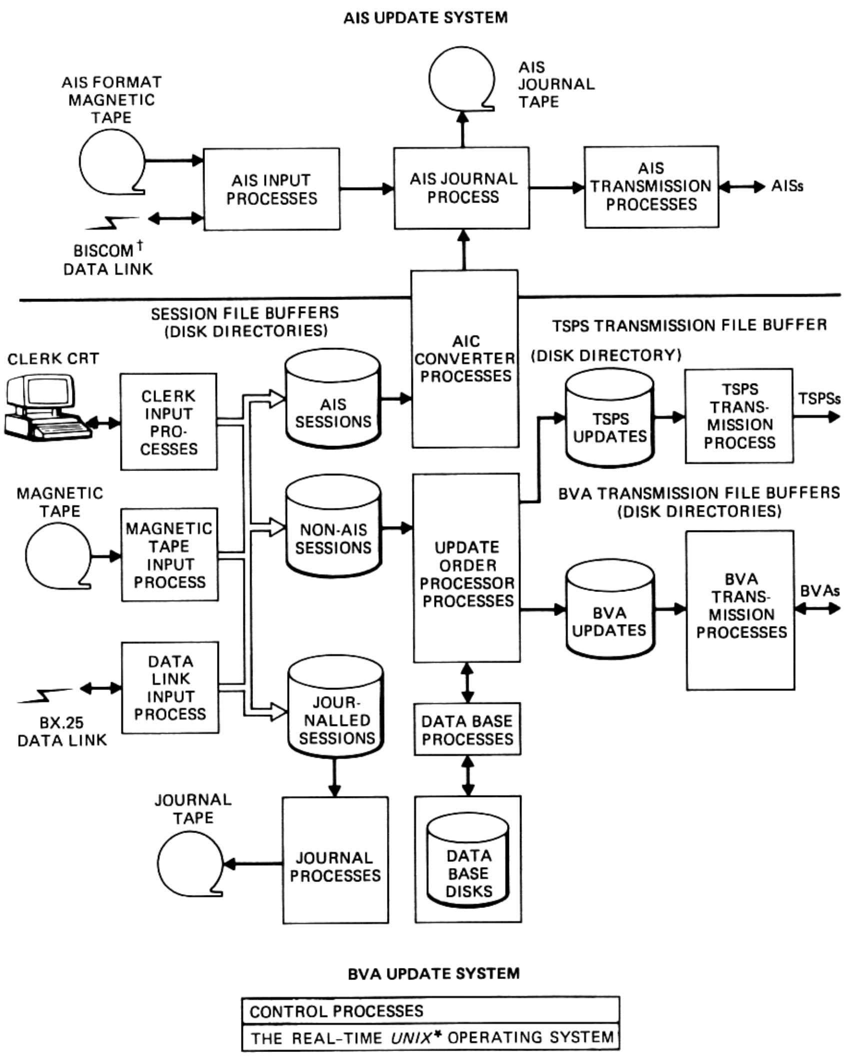
In Fig. 1, the software processes retained from the No. 1 DBAS are shown above the horizontal line. They will be referred to as the AIS Update System. The new software processes used for BVA and TSPS updating are shown below the line. They will be called the BVA Update System. The architecture is described in detail in Sections II and III.

1.3 The DBAS data base

When administering AIS data bases, the No. 1 DBAS was strictly a transaction-oriented system. Updates sent to the No. 1 DBAS by the telephone company's Service Order System (SOS) were passed directly to the AIS as soon as possible, after passing appropriate consistency checks. Although a copy of the total contents of the AIS data base was kept on tape at the File Administration Unit (FAU), which operated the No. 1 DBAS, no on-line accessible copy of the AIS data base was kept in DBAS. This meant that checking the exact current contents of the AIS data base required querying the AIS itself. Similarly, administrative reports based on the contents of the AIS data base were limited by the ability of the AIS to supply the required information.

For the large BVA data bases to be administered by the No. 2 DBAS, this type of administration would not be practical. Therefore, an online accessible data base was established on disk as part of the No. 2 DBAS. Further, this data base was designed to be the master copy of all data kept in the BVA. (No on-line copy of the AIS data base is maintained and the TSPS data base is an interim subset of the BVA data base.) Thus, all telephone company needs for information about

* *UNIX* is a trademark of Bell Laboratories.



†BUSINESS INFORMATION SYSTEMS—COMMUNICATIONS SYSTEM
**UNIX* IS A TRADEMARK OF BELL LABORATORIES

Fig. 1—DBAS software architecture.

data existing in the BVA data bases can be met by querying the DBAS. Therefore, the No. 2 DBAS was designed primarily as a data base system. Adequate throughput and integrity in data handling were required to be certain that the subset data bases kept in the BVAs were appropriate copies of the DBAS master data base.

The details of the DBAS data base are described in Section III.

II. THE DBAS SOFTWARE ARCHITECTURE

2.1 Throughput requirements

The DBAS is capable of handling the following throughput:

$$\tfrac{1}{2}\,\mathrm{DI} + \mathrm{MI} + \tfrac{1}{2}\,\mathrm{AU} + \mathrm{BU} > 7000/\text{hour},$$

where

$$\begin{aligned}\mathrm{DI} &= \text{number of data link inputs per hour}\\ \mathrm{MI} &= \text{number of clerk CRT (manual) inputs per hour}\\ \mathrm{AU} &= \text{number of AIS updates per hour}\\ \mathrm{BU} &= \text{number of BVA updates per hour.}\end{aligned}$$

For clerk and data link, an "input" is counted each time a line number is presented to DBAS. One "input" can result in several changes to data kept on a line. However, an "input" is still counted even if no change to line data results. Change data applied in special format to a "block" of lines is considered one input per line in the block. Inputs from magnetic tape are not considered in the formula as it is assumed that magnetic tapes can be read in during off-hours.

An AIS update is counted for each message sent by DBAS to an AIS. Here, blocks of line numbers count as only one update. A BVA update is counted if any change is sent to a BVA, or if the DBAS data base is accessed, even if no change is sent to a BVA. Blocks of lines again count as multiple updates if the DBAS data base changes.

In normal operation, most DBAS inputs will be mechanized (either data link or magnetic tape). Further, most transmissions to the BVA will be made during hours of light traffic at the BVA. However, the actual transmission to the BVA uses a small fraction of the total central processing unit (CPU) and disk time required per update.

2.2 Additional requirements

In addition to throughput requirements, the DBAS must meet several other design constraints:

(*i*) The AIS update latency must be low. When an AIS input has been completely received by DBAS, it is desirable for the update to be ready for transmission to the appropriate AIS within 10 minutes.

(*ii*) BVA update latency must be low. When a BVA "immediate" (high-priority) input has been completely entered into the DBAS, it is desirable for the update to be ready for transmission to the appropriate BVA within 10 minutes.

(*iii*) Clerk terminal response must be good. When clerk terminals are being used to enter data, they must be given priority over other processing tasks. This is because clerk terminals will generally be used only to enter high-priority input or to perform troubleshooting.

2.3 BVA Update System software architecture

As shown in Fig. 1, DBAS input can be received via either clerk entry, magnetic tape, or data link. A software process is associated with each input medium. For clerk input, a software process is associated with each physical CRT. These clerk input processes are separated, but share program text to minimize their memory utilization.

The role of each input process is first to deal with the specifics of each input medium. For example, the magnetic tape input process checks tape headers and trailers; the clerk input process interprets keystrokes and interacts with the clerk. All processes then perform basic syntax and consistency checks on the input they receive. These checks are the same for any input medium.

For each input process, a group of inputs is collected to form a "session." A session may contain up to 256 separate inputs. When all input for a session is complete, the entire session is written into a disk file under the *UNIX* operating system file system. This disk file may be linked into several directories for access by the appropriate processes. Inputs within a session are thereafter processed as a group, improving processing efficiency.

For example, every session file containing any error-free input is linked to a directory where it can be accessed by an Update Order Processor (UOP) process. The UOP reads the session file, and for each line number, retrieves the existing data (if any) for that line from the DBAS data base. The existing data are compared with the proposed changes to be sure the changes are reasonable. If so, the changes are made to the DBAS data base. If subsequent changes also need to be made to a BVA data base, the UOP creates a session file in a different directory accessible by the BVA transmission process.

With this architecture, significant buffering can be utilized at several points to load-balance and improve the total throughput possible on a daily basis. For example, consider a case where a substantial amount of clerk input was required in a particular hour. Work by the UOPs and data base on routine updates could effectively be suspended, and most of the CPU and disk resources could be available to clerk input processes. In practice, this is done by giving processes with terminal I/O work a higher priority in accessing the CPU. In this example, good clerk response time would be maintained. Sessions would be buffered in the directories between the clerk input processes and UOPs. Later, when the clerk input load was lower, the UOPs and data base software would begin processing the buffered routine updates. In general, this processing of buffered change data could continue for several hours past the conclusion of clerk data entry, if a peak in update volume was seen on a given day. This processing of buffered data would be completed before the scheduled time of data transmission to the BVAS, which occurs in the middle of the night.

2.4 Integrated input

Early in the design of the No. 2 DBAS, it was recognized that if changes affecting services provided by a BVA were requested on a line, changes would usually also be applied to an AIS. Therefore, an input to the BVA Update System would have to be duplicated as an input to the AIS Update System, if these two software systems were kept fully independent. This would be a very inefficient "double entry."

Therefore, a process called the Automatic Intercept Center (AIC) Converter was included in the software architecture. This process is shown crossing the center line on Fig. 1. If an input process in the BVA Update System finds input requiring an AIS update, it links the session file to a directory whose contents are scanned by the AIC Converter process. The AIC Converter reads the session file and extracts update information for the AISs. This information is converted from the data formats used in the BVA Update System to the format used in the AIS Update System. The update is then inserted into the AIS Update System's processing stream.

2.5 Performance-guided development

From the start of development of the No. 2 DBAS, it was recognized that the throughput requirements would use a substantial fraction of the available resources of a PDP 11/70 running the real-time *UNIX* operating system. Therefore, it was decided that the development of No. 2 DBAS would be guided by the need to meet the performance requirements. A first step in this direction was to develop a simulation of the No. 2 DBAS. This simulation was to prove whether or not the requirements could be met, and possibly suggest improvements in the software architecture.

Several different types of simulation were considered. These included analytic models, General-Purpose Simulation System (GPSS) simulations, etc. The method finally chosen was a functional process simulation, wherein some of the developers write model processes for all processes in the proposed architecture. These simulated processes then incorporate dummy processing loops, make disk reads and writes, and perform interprocess communication in an attempt to utilize system CPU and disk resources in the same way as the eventual software system.

A functional process simulation has several advantages. There is no need to simulate the hardware or operating system, as the "real thing" is being used. This eliminates considerable effort and removes substantial uncertainty from the simulation. The simulation can be evolved into the final product, recovering some of the effort required to produce the simulation. Since the simulation is written by developers of the final system, it gives them substantial exposure to the language, operating system, and project software architecture. This exposure can be especially valuable if the average level of experience of the designers is low. Finally, estimates of the amount of code and effort required for the final product can be improved by observing the development of the simulation.

For No. 2 DBAS, a functional process simulation of the proposed DBAS architecture was developed. As compared with Fig. 1, the initial architecture had the equivalent of a single UOP, accepting sessions from one session file buffer. No simulation was provided for the data link input capability, and no attempt was made to interface with actual AIS or BVA hardware.

Initial results of the simulation are shown in Table I. The simulation indicated that the proposed architecture, running on a PDP 11/70 under the real-time *UNIX* operating system, could meet the DBAS throughput requirements. However, when measurements were taken of CPU and disk utilization for the case of magnetic tape input (line 1 of Table I), both were substantially less than 100 percent. This meant further improvement could still be obtained.

Analysis revealed that a single UOP interacting with the data base tended to serialize the utilization of CPU and disk. For example, the UOP and data base processes would simultaneously block waiting for disk I/O, and the CPU utilization would drop during this interval. Therefore, more concurrency was required. The simulation was modified to include the equivalent of three UOPs operating on the same input session directory, each handling approximately every third session file. The throughput now became as shown in line 5 of Table I. Using this as a guide, the No. 2 DBAS BVA Update System software architecture was modified to the final version shown in Figs. 1 and 2. This architecture will be discussed in more detail in Section III.

In addition, analysis of results of the simulation contributed to selection of the clerk terminal, and helped in the human factors design

Table I—Data Base Administration System performance simulation—typical results with continuous magnetic tape input

Simulated Test Run	Clerk Inputs Per Hour	BVA Updates per Hour	AIC Updates per Hour	Total
1	(Tape only)	9391	1356	10,747
2	681	9388	1632	11,020
3	1406	8795	1535	10,330
4	2071	7020	1479	8499
5	(Tape only) improved architecture	13,299	1555	14,854

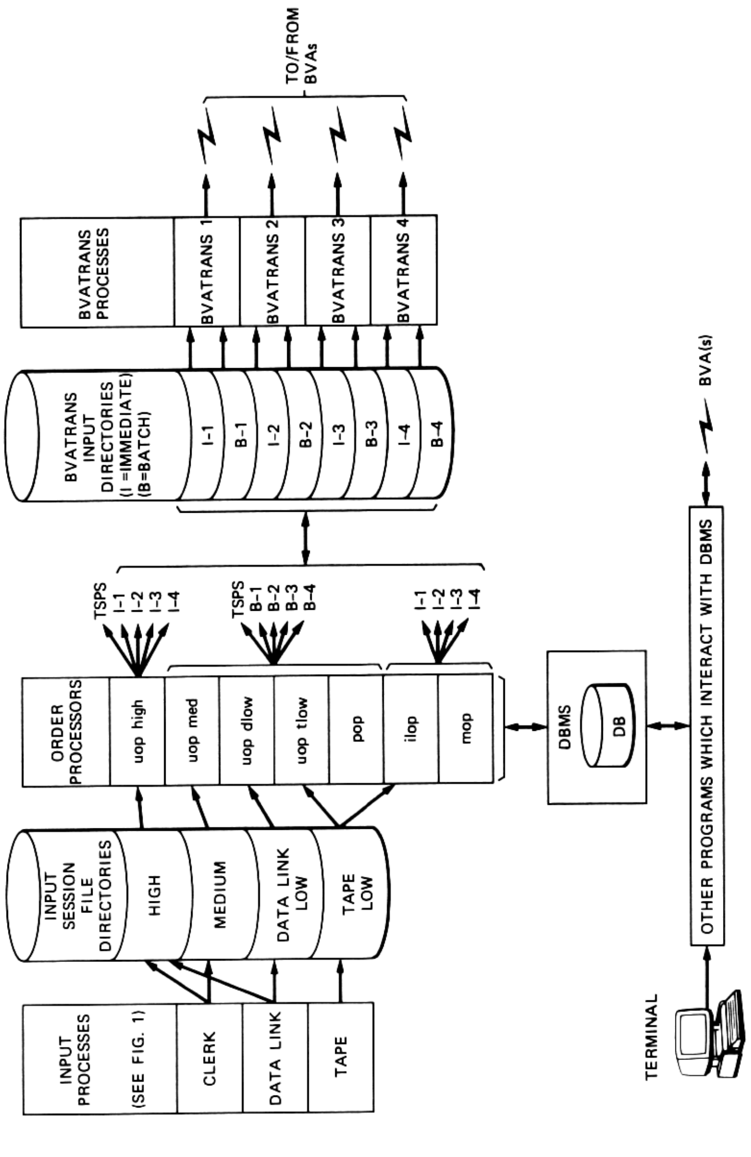


Fig. 2—Update processing architecture.

of the clerk interface. The original software written for the simulation proved to be of continuing value during development, in addition to being the basis for the final product. For example, during the design of the interprocess communication (see Section 2.6), a question arose as to the utilization of the real-time *UNIX* operating system message buffers. The simulation was run under heavy load with special message buffer instrumentation in the operating system, and was used to determine the utilization level.

As a result of this performance-guided development approach, the initial release of the No. 2 DBAS successfully met its performance objectives.

2.6 Process control and interprocess communication

When programs are run in the typical *UNIX* operating system time-share environment, they are usually associated with the CRT (or teletypewriter TTY) of a single user. Users and processes are very effectively prevented from interfering with each other, and each user and the user's programs see an entire "virtual" computer system.

However, for applications such as No. 2 DBAS, some degree of efficient process interaction is desirable, and it is also necessary to be able to control a group of processes as a software system. Both of these required additional development specific to DBAS.

In general, No. 2 DBAS processes are independent and modular. For example, it is possible to run a clerk input process from a *UNIX* operating system terminal like any other *UNIX* program. Session files can be created, and would be processed normally, if the rest of the DBAS software were subsequently run. However, there are important exceptions to this independence. For example, most administrative report programs can be run from *UNIX* terminals like the clerk input program. But, they cannot produce a report on the DBAS data base unless the data base processes have previously been properly started. These interdependencies between processes could be confusing to the DBAS user. To establish the complete BVA Update System, more than a dozen separate processes would have to be started. During operation, most of these processes produce brief reports on their progress, which would result in scrambled printouts if all were directed arbitrarily to a common terminal.

To solve this, a set of processes was designed, collectively called CONTROL. The DBAS user starts CONTROL from an arbitrarily designated control terminal, specifying one of several modes of operation. Based on this mode, CONTROL creates and runs all the processes that the user requires. For example, in the "update" mode, all programs needed for BVA updating are started. In the "data base" mode, only the data base processes are started. The user can then run various administrative programs successfully, and can change critical per-site data with the knowledge that no updates (which might depend on the changing data) are in progress.

CONTROL also handles the orderly demise of processes it has started. If ordered by the user to perform a "graceful shutdown," CONTROL requests each process it has started to exit at the next convenient stopping point (e.g., completion of processing the current session file). If the user requests an "immediate shutdown," CONTROL requires each process it has started to quit immediately. This request is only made under rare circumstances.

All processes running under CONTROL have access (through CONTROL) to the terminal at which CONTROL is running. They can print messages to the user, or request user input. Therefore, this single (control) terminal can communicate with any desired process, and messages output by any process under CONTROL are printed in an orderly, meaningful manner, where the process requesting the printout is clearly identified. This communication capability is provided by a standard interprocess communication package based upon the real-time *UNIX* operating system "message" feature.

CONTROL can also start processes at a scheduled time of day. For example, if transmission of routine updates to a BVA has been scheduled for 11:00 p.m. daily, CONTROL will automatically start the appropriate processes at that time. Several such events can be scheduled by the DBAS users via administrative programs.

III. THE DBAS DATA BASE MANAGEMENT SYSTEM, ORDER PROCESSING, AND BVA TRANSMISSION PROGRAMS

The DBAS Data Base Management System (DBMS), Order Processing, and BVA transmission programs form a group of interrelated programs that were designed to provide three specific functions:

(*i*) The initial load of the DBAS and the BVA Data Bases.

(*ii*) The ongoing daily update capabilities of DBAS and BVAS.

(*iii*) Data base house cleaning, backup, and restoration functions.

Each of these functions is described in detail below with an accompanying explanation of the contribution of their components.

3.1 Initial load function

An initial load starts with tape input of billing number data. The DBAS stores these data in its data base (DB) and then immediately sends a copy to its associated BVAS. If run continuously, the initial load of even the largest DBAS DB and its BVAS could be completed in about one week. However, the initial load might have to be interrupted to process normal updates for connected AISS or TSPSS. For example, the DBAS might be used for initial loading on the weekend, but during the week, it would be used to process normal updates. In this case, subsequent weekends are necessary to initially load the DBAS and BVA data bases until both data bases contain all of the line numbers assigned to that DBAS.

The important point here is that initial loading is done at a different time and quite separately from regular daily updating. This method of operation arises from the need to process updates at considerably different speeds in each case.

The initial loading rate per update must be much faster than the daily update rate because the DBAS data base may contain from 1 to 12 million records (average approximately 4 million records) that have to be initially loaded, but only about 1 percent of the total records are modified per day. Thus, the normal update rate of DBAS can be much slower than the initial load rate. The normal update rate capability of DBAS is about 7,000 updates per hour as mentioned in Section 2.1. In contrast, an initial load rate of over 100,000 updates per hour was achieved. This required the following special developments:

(*i*) An initial load tape format was specified requiring all billing numbers in a billing number group (BNG) to be sequentially ordered on the input tape to DBAS. This avoids a sorting operation normally performed during routine updating.

(*ii*) An Initial Load Order Processor (ILOP) was developed which ensures the sequentiality and uniqueness of each set of updates in a billing number group. It also generates files for the DBAS data base and BVA transmission programs.

(*iii*) An Initial Load Data Base Management System (ILDBMS) was developed to store complete BNG files on contiguous strips of DBAS disks.

The ILOP and ILDBMS are shown in Fig. 3, along with the other programs involved in initial load processing. The READTAPE and BVATRANS programs are also used for normal daily updating. The common use of these programs was a design goal aimed at minimizing the number of different DBAS programs that had to be developed, tested, and maintained. As a result, the unique capabilities needed for initial loading were implemented entirely in the ILOP and the ILDBMS.

The ILOP and ILDBMS were designed to work very closely together. The set of all updates in each BNG is in order of line number. The resultant file is then passed to ILDBMS via a "store-all" command used only for initial load. The ILDBMS reads each update, assigns them to buckets (disk pages) and writes the subsequent complete set of buckets into contiguous disk blocks belonging to the DBMS. This marriage of functions and strip loading of DB disk space basically underlies DBAS's relatively fast initial load rate.

The BVA transmission program (BVATRANS) gets its input files from

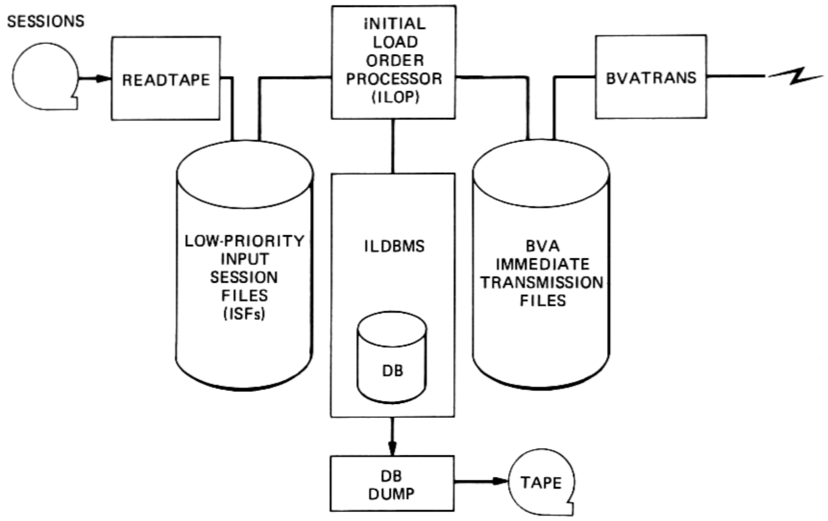

SESSIONS
READTAPE
INITIAL LOAD ORDER PROCESSOR (ILOP)
BVATRANS
LOW-PRIORITY INPUT SESSION FILES (ISFs)
ILDBMS
DB
BVA IMMEDIATE TRANSMISSION FILES
DB DUMP
TAPE

Fig. 3—Initial load architecture.

the ILOP and transmits the file immediately to the BVA. Since this file is sequentially ordered by line numbers (directly from the input) the data can be packed compactly into the packets that DBAS uses to communicate with its BVAS. The BX.25 protocol is used for this communication. Each packet contains billing number records for only one billing number group. Up to 126 billing number records may be sent in a packet. Since there are potentially several thousand billing number records in every billing number group in the initial load stage, the packets are usually filled to capacity and the resultant transmission between DBAS and the BVA is very efficient.

Procedures for accomplishing the initial load were distributed to the telephone companies. These specified how various initial load situations should be handled. In particular, the situations of concern were the conversion of early DBAS generics to 2DB3, which is the latest generic. For example, if a company was using a 2DB2 generic, its DBAS would contain a data base consisting of any data used to load certain TSPS data. It was necessary to develop the program called DBDUMP shown in Fig. 3 by which the 2DB2 data base was dumped on a tape in a format that was compatible with inputting to a 2DB3 system. Since the 2DB2 data bases are generally small (less than 100,000 records) the time to reinsert the 2DB2 records was not of concern.

It was necessary to retain the 2DB2 data during and sometime after the 2DB3 initial load because telephone companies may have to continue updating TSPS data bases until the BVA(s) are activated. Other potential generic conversion situations were explored and documented for the use of initial load coordinators in each telephone company.

3.2 Daily update processes

After the initial load for each BNG is completed, updates to these BNGS are entered using clerk, data link, and/or tape input medium. Each corresponding input program groups these updates into Input Session Files (ISFS). The input process then links the ISFS that are formed to one of three types of directories: high, medium, or low priority. Priority has the following significance:

(*i*) High-priority sessions contain updates that must be transmitted to the BVAS immediately (i.e., within 10 minutes). Any type of update can be input at high priority.

(*ii*) Medium and low priorities are intended for ISFS containing updates that do not have to be transmitted to BVAS until a specified time of day, usually in the evening. Most updates will be of this type. For example, only a few hundred high-priority updates per day would usually be entered as compared to about 30,000 medium- or low-priority updates.

As in the initial load architecture, order processors read ISFS, format and enter changes to the DBMS, and generate files for transmission to BVAS; the DBMS stores records; and BVATRANS sends the updates to the BVA. However, in contrast to the initial load situation, ISFS in the update mode are generated by three types of input media, and the update data will generally be randomly ordered (as opposed to sequentially ordered in the initial load mode). This required setting up a more complex order processor architecture as shown in Fig. 2. An Update Order Processor (UOP) is associated with each type of input directory. The input session files generated by each of the input processes are very similar, so that a single program was designed to process all the various types of ISFS as described below.

All UOPS are started by CONTROL and normally run all day. When created, each UOP is passed a parameter indicating its source of ISFS. This parameter also sets up each UOP to modify its operation as follows: The UOP which processes high-priority ISFS links its output to immediate BVATRANS input directories (e.g., to I-1, I-2, etc.); the UOPS processing lower priority ISFS link their output to the batch input directories of the BVATRANS processes (e.g., to B-1, B-2, etc.). Otherwise, all UOPS handle ISFS in the same manner—when first started, the UOP creates a BVA update file for each BVA in its own data directories. Then, whenever a UOP completes processing an ISF, it determines which BVA update files have been used and sends them to the appropriate BVATRANS. That is, it closes the file, links it to the BVATRANS data directory, and creates a new BVA update file so that it can continue processing. Each UOP performs various checks on every update in the ISF to ensure that the new data are consistent with the data already in the DBAS DB prior to finally updating the DBAS data base, the BVAS, and the TSPSs. Checks that fail result in exception reports describing the reason for the report and the input and data base records in question. For example, an ISF update specifying a Personal Identification Number (PIN) for a public telephone record in the DB would be rejected with an appropriate message.

The Pending Order Processor (POP) does not run continuously like the UOPs. Instead, it runs briefly once a day to process the pending orders in the data base whose effective dates become current. A pending order is any update whose effective date is tomorrow or up to 6 months further in the future. The POP obtains a list of pending updates from the DB, modifies the DBAS DB, and then produces a single BVA update file for each BVA linking those files to the appropriate BVATRANS batch data directories.

The Move Order Processor (MOP) is started by the DBAS administrator when it is necessary to move the data stored for a set of BNGS from one BVA to another. This process obtains a file from the DBMS for an entire BNG which has been specified as a parameter in the DBAS move command, reformats the file, and links it to the target BVATRANS batch directory. If more than one BNG is required, the program repeats the above for each.

Figure 2 also shows that multiple instances of BVATRANS are used to communicate with a corresponding number of BVAS. A parameter is passed by CONTROL when it creates each BVATRANS process. This parameter tells each BVATRANS process which input directories to use for its corresponding BVA. It should be noted that several BVATRANS processes might be invoked in the same manner for initial loading several BVAS simultaneously.

3.2.1 Handling mixed priority and multiple updates

The DBAS is required to process and output multiple updates for billing numbers (BNS) in the same sequence as they were entered in any given day from a given source (tape, clerk, or data link). Furthermore, a high-priority update for a particular BN must override any previous lower priority update for that BN which may have been processed the same day. For example, a high-priority service denial update should block any lower priority change order which may have been on that day's tape and processed earlier. These requirements were implemented in the following manner.

When any medium- or low-priority UOP reads an update from its ISF, the corresponding DB record is also retrieved. The date when the DB record was last changed is compared to the current date (which it gets from the ISF name). If the two are the same, and the DB record was last changed by a high-priority order, then the new update is rejected with an appropriate exception report. If the DB record was last changed by a low-priority order, the UOP time stamps each update passed to BVATRANS. The new update and the time stamp are passed along to the appropriate BVATRANS. In its batching mode, BVATRANS sorts updates by line number and then by time stamp to properly order multiple line number entries. A high-priority update for a BN that was preceded by a lower priority update that same day is "marked" by the UOP. This mark permits BVATRANS to block the latter updates by an in-core filter routine.

3.2.2 The DBAS data base management system (DBAS DBMS)

The major challenges in the DBAS DBMS design were its size and its high update volume: up to 12 million records with up to 100,000 updates per day. To meet these needs, the most interesting features are briefly described below.

(*i*) File structure and file access—The data base comprises a set of *UNIX* operating system special files accessed by raw I/O, each being a DEC RP06 disk pack (176 Mb). A two-stage extendible hashing algorithm is used to access the record of a given ten-digit billing number.[2] The number of records in the data base can grow and shrink dynamically. The record size (10 to 50 bytes) is variable. The average number of disk accesses per update is four to six.

(*ii*) Concurrency control—An Application Program (AP) is either an order processor or an administrative process outside the DBMS. Multiple APs can access the data base at the same time to make the system easier to use. Multiple copies of the lower level DB access modules can be run at the same time to achieve a high data rate between core memory and secondary storage devices. This is achieved through the implementation of a two-level hierarchical locking scheme. An AP can either lock the whole data base or a portion of the data base. Only exclusive locks are available for simplicity. Deadlock is avoided by the restriction that each AP can request and hold at most one lock at a time.

(*iii*) Data independence—Data base conceptual schema and views are provided at the lower access level, and the APs, for example the UOPs, have direct access to these lower access modules. User process security is enforced by the restriction that each AP sees only the data it needs through a predefined view.

(*iv*) Buffer management—A large number of buffers can reduce the number of data base disk accesses. The *UNIX* operating system on the PDP 11/70 restricts the size of a particular process's virtual address space to 128 kb. Compared with a 2-Mb physical core memory, this small virtual address space presents a problem in buffer management design. The problem was solved by sharing a set of buffers between the virtual address space of the lower level access module and physical main memory. When a data base replacement request follows an earlier retrieval request, no additional disk accesses are required to reference the same data base record in core memory.

(*v*) Secondary storage management—Contiguous disk blocks are freed and allocated dynamically. The disk write module writes one or more contiguous disk blocks from contiguous buffer space in a single real-time *UNIX* operating system I/O request. When contiguous disk space is available, these features reduce the number of disk I/O calls.

(*vi*) Secondary key retrieval—A restricted form of secondary key retrieval is provided to handle the pending service order feature, which permits inputting service orders to be processed at a future date. A file of record keys (each representing a record with a pending order for a given date) is output by the lower level access modules using the date as the secondary key.

3.2.3 Internal operation of the Data Base Management System (DBMS)

A process view of the DBMS is shown in Fig. 4. This shows that the system comprises two main processes: the Data Base Manager (DBM) and the Access Task Process (ATP). Features described above in Section 3.2.2 are implemented in these two processes which operate as follows: Application Programs initially interact with the data base via "opendb" or "begin session" commands. The DBM assigns the ATP to the process, keeps track of views in use, and provides for recovery as described in the next section. Upon receiving an acknowledgment from the DBM to its initial command, the AP accesses the data base through another set of commands now directed to the ATP, such as "retrieve," "replace," "delete," or "store" data.

The set of all commands that are available for APs to access the data base is called the Data Manipulation Language (DML). The commands were incorporated as subroutine calls in each AP, and the subroutines actually interact with either the DBM or ATP. The subroutines also convert the data for the DBMS. For example, line numbers are converted to long integers. Hence, the DBAS DBMS is a special-purpose DBMS in that only special values of some data fields are allowed. In this manner, data are stored quite compactly in DBMS disk pages (over a hundred records per page).

3.3 The DBAS backup and recovery

The DBAS data base is backed up by periodic disk-to-disk copying of the data base disks. Following certain severe system failures it may

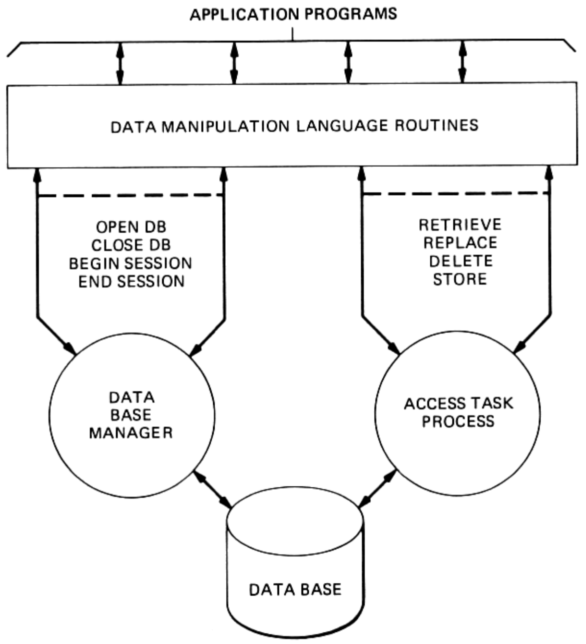


Fig. 4—Data Base Management System.

become necessary to recreate the DBAS data base, starting from these backup copies and integrating updates received between the date that the backup copies were made and the time that the failure occurred.

During normal updating operations, a special DBMS "checkpoint" mechanism is employed to protect the integrity and consistency of the DB in the event of system crashes or other system stoppages. The explanation of this vital feature is as follows: The data base disks are separated into read-only disks and writable disks. The primary data structure of the data base is a tree of disk blocks. The blocks of the first two levels, the root and its children blocks, reside in core memory to minimize the number of disk accesses per data base record access. Whenever a block on a read-only disk (a write-protected disk drive) is to be modified, writable disk blocks are allocated on a separate writable disk known as the "working volume." One or more of the newly allocated blocks are then written with the modified data, as well as with its ancestor blocks (except the first two levels) so that the contents of read-only blocks are not touched. Disk blocks on the writable disk belonging to the most recent consistent data base copy are temporarily assigned read-only status until a new consistent copy becomes available. At the end of each day, the writable disks and the read-only disks are merged to produce a new set of read-only disks for the next day.

For this purpose, a Merge DBMS was designed whose only functions are merging and some daily DB housecleaning. Also, at regular time intervals during the day, root blocks are written to save a consistent copy of the data base on disk. This is called the data base checkpoint. Before the checkpoint is taken, the system synchronizes itself, through blocking, so that no updates are in progress to ensure that data are consistent.

In case of accidental machine failure, only the work done since the previous checkpoint needs to be repeated, thus permitting easier and faster recovery. Following a successful merge, the old working volume is saved, along with the most recent backup copy of the data base disks. They may be reused at the same time as the associated data base backup disks, usually in two weeks. To facilitate system recovery, the DBMS maintains a session trace file for the sessions that UOPs have processed. For each session, all UOPs send Session-Begin (SB) and Session-End (SE) messages immediately before and after processing data in their session files. The DBM keeps the time of these messages and the session identification in the trace file. When a checkpoint is to be taken, the DBM stops replying to SB messages, thereby halting the UOPs when they reach an SB. Programs other than UOPs are allowed to finish their processing, but no new ones may start while the checkpoint is pending. When all writing has been stopped, the DBM does its checkpoint routines. It then resumes replies to SB messages so that all application programs can continue.

A necessary condition in restarting after a machine failure is for the APs and DBM to have a consistent view of the data base. Upon a crash or other ungraceful system stoppages, a special program is invoked to examine the session trace file so as to identify successfully completed sessions. Damaged or missing sessions need to be reinput from the original input medium or from the journal tape. As mentioned in a previous article, a journal tape is maintained by the DBAS journal program that can keep a record of all service order inputs. The DBAC (Data Base Administration Center) operator must notify those responsible for inputting data, if some sessions have to be repeated. It should be noted that specially tailored checkpoint capabilities are provided in both the Initial Load and Merge DBMSs. Hence, these systems can recover from certain common system stoppages without having to reprocess much data just like the regular DBMS.

IV. SUMMARY

A performance-guided analysis of required features and operating environment led to an early choice of the DBAS's architecture. The resultant design has the following capabilities: a wide range of administrative, control, order processing, data base, and BVA interaction functions; initial load rates of 100,000 updates per hour; ongoing update rates of 7,000 updates per hour; a special-purpose, yet flexible, Data Base Management System; and an innovative backup and recovery scheme that permits the use of a simplex processor and simplex periphery.

V. ACKNOWLEDGMENTS

Many designers and testers contributed to the success of DBAS. For the development described in this paper, the authors especially want to acknowledge the members of Bell Laboratories Operational Software Design Department.

REFERENCES

1. "*UNIX* Time Sharing System," *Bell System Technical Journal, 57,* No. 6, Part 2 (July–August, 1978).
2. R. Fagin, et al., "Extendible Hashing—A Fast Access Method for Dynamic Files," ACM Transactions Data Base System, *4* (1979), pp. 315–44.