CM-P00063113

June 7, 1991

The L3 Database System

B. Adeva,⁶ P. Bagnaia,¹² S. Banerjee,³ L. Barone,¹² D. Boutigny,²
 F. Bruyant,⁶ P. Cardenal,⁶ N. Colino,⁶ E. Gonzalez,⁸ M. Guanziroli,¹³
 Y. Karyotakis,^{2,6} F. Linde,¹¹ R. Mount,¹⁰ E. Nagy,⁴ L. Niessen,¹ J. Perrier,⁷
 M. Pieri,⁵ J. Rose,¹ S. Shevchenko,⁹ A.A. Syed,¹³ I. Vorobiev,⁹ Y. F. Wang⁵

Abstract

The database system developed for the L3 experiment at the CERN Large Electron-Positron collider is described. The solution chosen has two components: a FORTRAN package (DBL3) and a system of distributed servers and data (L3DBSM). DBL3 provides basic database functions supplemented by facilities specific to high energy physics experiments. The L3DBSM system, operational since the start of LEP data taking, has handled efficiently several hundred Mbytes of data over three different systems (VAX/VMS, IBM VM/CMS, Apollo Aegis SR10.x) and at eight different sites.

(Submitted to Nuclear Instruments and Methods in Physics Research)

-
1. I. Physikalisches Institut, RWTH, Aachen, Federal Republic of Germany[§]
 - III. Physikalisches Institut, RWTH, Aachen, Federal Republic of Germany[§]
 2. Laboratoire de Physique des Particules, LAPP, Annecy, France
 3. Tata Institute of Fundamental Research, Bombay, India
 4. Central Research Institute for Physics of the Hungarian Academy of Sciences, Budapest, Hungary
 5. INFN Sezione di Firenze and University of Firenze, Italy
 6. European Laboratory for Particle Physics, CERN, Geneva, Switzerland
 7. University of Geneva, Geneva, Switzerland
 8. Centro de Investigaciones Energeticas, Medioambientales y Tecnologicas, CIEMAT, Madrid, Spain
 9. Institute of Theoretical and Experimental Physics, ITEP, Moscow, Soviet Union
 10. California Institute of Technology, Pasadena, United States of America
 11. Carnegie Mellon University, Pittsburgh, United States of America
 12. INFN-Sezione di Roma and University of Roma, "La Sapienza", Italy
 13. World Laboratory, FBLJA Project, Geneva, Switzerland
- [§] Supported by the German Bundesministerium für Forschung und Technologie

1 Introduction

The L3 detector at LEP consists of a central tracking chamber, an electromagnetic calorimeter, a ring of scintillation counters, a hadron calorimeter, and a muon chamber system [1]. Two forward calorimeters are used to measure luminosity. The elements of the L3 detector were designed to measure electrons, muons and photons with high precision. However, the intrinsic resolution of the apparatus could only be realised through continuous monitoring and calibration.

The L3 collaboration is composed of institutions from 35 different countries. Although data processing is centralized at CERN, systematic studies of detector performance and reprocessing of the data are usually done at outside centers. This requires availability of monitoring and calibration constants on different computers in different places, with minimum time delay, and total consistency.

A database system has been designed and successfully implemented to meet these requirements. This system has been operational since the start of LEP data taking (August 1989), and has handled efficiently 150 Mbytes of data in 1989 and 400 Mbytes of data in 1990, over three different systems (VAX/VMS, IBM VM/CMS, Apollo Aegis SR10.x) and at 8 different sites (CERN, Geneva, Bologna, Madrid, Nijmegen, Aachen, Zürich, and Lyon).

The general requirements of the experiment and an overview of the database system are presented in sections 2 and 3, respectively. The content of the database is described in section 4. The transfer of monitoring and calibration data to the online database is outlined in section 5. Sections 6 and 7 describe the DBL3 package and the L3DBSM system. The performance of the system, the problems we encountered in operating it, and possible improvements with a view to similar applications for large and complex detectors at future colliders (LHC/SSC) are discussed in section 8.

2 General System Requirements

The L3 database management system has been designed to meet the following requirements of the experiment:

1. Calibration and monitoring data, mostly produced online, must be available within an hour on the offline computer in order to process data as quickly as possible. It is desirable to have these data available within a few hours on other sites for calibration studies and refits.
2. Users must have the possibility of running the same code on every computer, irrespective of the environment (online or offline) and of the development phase of their programs (tests with a private database or standard use).
3. Integrity of data is essential. The system must ensure that neither users by accident, nor computer crashes can affect the content of the database files.

4. The system must be operated automatically as far as possible, and human intervention should be triggered by the programs in case of emergency.
5. The system must be able to handle efficiently calibration data, information on data taking and data processing, as well as luminosity data and all information relevant to the LEP machine.

3 Outline of the Database System

The solution chosen has two components: a FORTRAN package with basic database functions supplemented by facilities specific to high energy physics (HEP) experiments (the DBL3 package) and a system of distributed servers and data (the L3DBSM system).

The DBL3 package, written within the collaboration, has been built on top of the RZ direct access package [2]. This choice was quite natural, since the L3 software relies heavily on the ZEBRA data structure management system. The L3DBSM system has been designed as a client-server scheme, where we define the *client* as any program which needs to modify the database, and the *server* as the automatic program which interacts with the clients, handles their requests, and physically modifies the database. An overview of the system is shown in Figure 1. Servers exchange database updates by means of sequential files called *journal files*. Journal files serve the additional purpose of enabling recovery in the event of database corruption.

Two other choices have been made in order to simplify the management of the system:

1. Each subdetector, including trigger and run/job control, has a separate database file, leading to a total of eight files.
2. The database files on the LEPICS computer (an IBM 3090/180) are considered *master databases*. This means that the databases on other offline computers are only updated on receipt of journal files from LEPICS.

Commercial products were examined at the start of the L3DBSM/DBL3 development [3,4,5], but were not judged suitable for the needs of the experiment because of high cost and the need for additional software to fulfill specific requirements of the experiment.

4 Content of database

The database files contain all the necessary information required for a precise description of the status of the detector and of its electronics at any time during data acquisition, as well as for book-keeping of data processing and Monte Carlo event simulation.

It is convenient to class the information that describes the status of the detector according to the frequency with which it is updated.

1. *Fixed data:* These include all parameters measured during the construction of the detector such as geometrical parameters and some calibration constants. It includes, for example, in the case of the BGO calorimeter, individual crystal dimensions, amplifier gains, light response curves, energy calibration constants, etc. These parameters have been measured before installation in the experimental area, either in the laboratory or in test beams. They occupy a small amount of space.
2. *Slowly varying data:* Knowledge of the status of the detector is essential to the offline reconstruction. Therefore, lists of dead or noisy detector elements and electronic channels are kept up-to-date. As breakdowns of detector elements or electronic modules are rare, the volume filled by this information grows at a very slow rate.

Each subdetector contains a large number of electronic channels, from 2000 for the vertex detector to 27000 for the hadron calorimeter. A description of the electronic chains, from detector elements to preamplifiers, shapers, ADCs or TDCs, and readout modules, is stored in the database. To ease the location of electronic modules and to allow a quick traceback in case of module malfunction, their positions in crates and racks are also recorded in the database.

Other slowly increasing data include algorithms and parameters of the trigger system and of the microprocessors that treat subdetector data, help files, and logbooks.

3. *Periodic calibrations:* The conversion of raw data from ADC or TDC units to energy or drift distance is done offline. To achieve the best possible resolution the characteristics of each individual electronic channel are taken into account. These characteristics (pedestal, gain, linearity, delay, noise level) may vary with time. They are measured daily and stored in the database. Other calibrations, *e.g.* checks of the energy scale of the hadron calorimeter, are performed on a weekly or monthly basis.
4. *Continuously monitored parameters:* The response and resolution of the detector depend upon parameters that cannot be controlled precisely such as the temperature, the pressure and composition of the gas (vertex detector and muon chambers), the value of the magnetic field, and the high voltage settings. These parameters constitute about half the data stored online in the database.

The amount of space taken up by the different classes of information are given in Table 1 for the 1990 running period (160 days). In total, the precise description of the detector required 247 Mbytes. Note that data produced by the periodic calibrations and by the continuous monitoring of parameters (96% of the database volume) are needed only for reconstruction at the subdetector level, and therefore do not need to be available for the later stages of physics analysis. As an example of a subdetector database, the muon chamber database structure is shown in Figure 2a. The structure of the subdetector databases keeps data of different types in separate areas which allows reduced databases to be produced easily.

Offline analysis of subdetectors, based on real events or special calibration runs, lead to corrections to initial subdetector calibrations. These new calibrations are applied during a second reconstruction of all events.

Part of the database (158 Mbytes) is dedicated to the book-keeping of the offline data processing and Monte Carlo event generation. It includes precise specifications of the programs and their conditions of utilisation, as well as the performance and status of all production jobs.

5 Production and transfer of online monitoring data

The paths followed by the data, from data producers to servers, vary widely with subdetectors. However, similar approaches have been adopted for similar types of data. In each case, the goal is to ensure a transfer that is safe even if the database server, the computer, or the network crashes. In addition, consistency checks of the data and validation by the user must be allowed before storage.

The database directory structures are created by stand-alone programs that may, at the same time, store part of the fixed data. The rest of the fixed data and the initial values of the slowly varying data are stored and updated in two different ways:

1. The local editor is used to display and update the data. Subsequently, a stand-alone program checks the input data and stores them in the database. Data checks include verification of type, range, and consistency.
2. Dedicated interactive programs, most of them based on the MODEL package [6] or on the L3 Online Window Manager Facility [7], present menus that follow closely the tree structure of the database. At each level, ad-hoc panels allow the user to examine and update the data. Data are checked whenever a panel is filled. These programs are user-friendly and can be used by a non-specialist but require more maintenance.

Most online calibrations are carried out by stand-alone programs which collect and analyse the data, present the results to the user, and store them in the database immediately after validation. Calibration results obtained on other computers and required on the online cluster are transferred and stored in the database by the same programs that treat the slowly varying data.

Monitoring of some parameters is critical not only to the offline reconstruction but also to the safe operation of the detector. For this reason, some monitoring tasks run independently of the data acquisition system, usually on dedicated computers. The monitoring systems, which do not know the status of the database servers and of the online cluster, must ensure a safe transfer of data to the online computer. As an example, the monitoring of muon chamber parameters is described below.

The muon chambers are continuously monitored by a network of microcomputers based on VME and controlled by the OS9/68k operating system [8]. The system works

in a completely automated way, although supervised by the person on shift who can modify its behaviour. The condition for sending new data to the database server depends on the nature of the data. Continuous variables, such as temperature, are sent whenever the difference from the previous reading is greater than half the magnitude that would introduce a significant correction in the reconstruction or would be dangerous to the detector. If the data describe a discrete state, such as the status of power supplies, any change of status triggers a transfer. The data collected by the VME microcomputers must travel through several networks before they reach their final destination. Data collected by the VME crates that read sensors travel through the OS9 network to one of the VME bridges to DecNet. The VME bridge sends the data to the online cluster where they are received by the database server. Two backup files ensure a safe transfer through the OS9 network. The data are saved locally as binary random access files on the original VME crate (first backup). On each VME bridge a server program polls these files and keeps records of all requests of data transfer (second backup). Finally another server in the same bridge reads the records and sends the data to an associated server on the online cluster using DecNet interprocess communication utilities. Once on the online cluster the data are sent simultaneously to the online database server and to the muon chamber online display system. In this way the person on shift can check the data being sent and the rate at which they are generated. Internal checks are done on each one of the previous steps. If a process finds a problem it sends a message to the person on shift and makes an entry in an appropriate logbook. This system, designed with data integrity in mind, allows data to be stored in the database a few minutes after their production.

6 The DBL3 Package

6.1 Functional Requirements

The database system has to support frequent calibrations made at arbitrary times, and to support access to the improved, but still highly time-dependent, calibrations that would become available for many detectors long after the data have been acquired. Access to these calibration data at any stage in the analysis has to be adequately efficient and to require that the user knows only the name of the calibration required, and the data acquisition time for which valid information is needed.

These goals, together with the constraints outlined in section 2, gave rise to the following specific requirements for the DBL3 package:

1. Database objects are data identifiable by a name and by two time parameters:
 - a hierarchically structured name similar to a UNIX pathname (an example is shown in Figure 2a). Optionally the pathname might point to a set of objects which could be distinguished individually by 32-bit 'keys',
 - a *period* of time for which the information in the object is considered valid (validity period),

- the *instant* of time at which the object was inserted into the database (insertion time).

Given the name of an object and the instant of time for which valid data are needed, the database should give access to the most recently inserted object of that name valid at that time. This retrieval process is shown in Figure 3.

2. With the exception of pathological cases, logical deletion of objects should occur by their being superseded by a more recently inserted object. By ignoring all objects inserted after a particular time, the database system should be able to make the database appear to be exactly as it was just before that time.
3. Access to database objects should be efficient even if the access were arbitrarily frequent. Thus the database system should manage a memory cache of recently accessed data.
4. Insertion of a calibration data object differing only slightly from an earlier object should not be discouraged. Thus data compression, such as the storage of only the differences between a newly inserted object and an earlier one, should be available and normally transparent.
5. Support for the journaling system described above should be provided.
6. It should be possible to differentiate between compact data, which could easily reside in memory, and bulky data which should stay on mass storage unless explicitly requested. Thus the system should provide efficient access to data composed of many small objects (such as the book-keeping data).
7. Although principally designed to provide database services to programs, the package also should have an interactive interface to allow knowledgeable users to query the content of the database.
8. The system should be readily implemented on the wide range of computers used by an experimental HEP collaboration.

6.2 Choice of Access Method or Database Package

In 1985 an evaluation was made of two commercial database systems and two random access packages developed within the HEP community [3,5]. Part of the evaluation consisted of the implementation of a simple model calibration database system followed by performance measurements.

The conclusions of this evaluation, which we believe may still be valid, were that, whatever the system chosen, substantial additional software would be needed to provide the functions described above. Commercial database systems would have reduced the effort required to ensure database integrity and would have offered good interactive interfaces. On the negative side, the performance of commercial systems was, at best, close to that of the HEP packages, and the problems associated with their purchase

and installation on a worldwide system of heterogeneous computers appeared to be insurmountable.

The HEP data structure manager ZEBRA [2,9,10] had already been chosen as a key component of L3 software which is written in FORTRAN. ZEBRA comprises several packages such as MZ, for data structure management in memory, FZ for sequential machine independent I/O, and RZ for random access I/O. It was natural to choose the RZ package as the access method for the DBL3 database system.

6.2.1 Functions provided by the RZ Package

This section reviews those functions of the RZ package which are important for DBL3. A more complete description of RZ can be found in the ZEBRA User Guide [2].

RZ data objects are stored and retrieved from directories in a hierarchical structure similar to the UNIX file system. RZ allows simultaneous access to an arbitrary number of physical random access files, each of which is seen by the application as a different top level directory. In principle, each RZ directory may contain a mixture of objects and other directories. In building up the L3 database, we have tried to minimize user confusion by storing data objects only in directories which are end nodes of the hierarchical structure. Each object has two components, one of which is easily scanned by the RZ package or by the application to select the desired object(s).

The first component of an object is its 'keys'. This is a vector of 32-bit words having the same length of up to 100 words for each object in a directory. The array of key vectors is the principal component of a directory and is normally cached in memory by ZEBRA whenever the directory is in use. The data type and name of each key is also stored by RZ.

The second and optional component of an object is its 'data'. The data for each object may be of any size, including zero, and are not cached in memory by RZ (unless they happen to lie inside the current physical disk I/O buffer). Although the data may be completely arbitrary, a very valuable facility for ZEBRA based applications is the ability to store ZEBRA data structures as RZ data.

The basic function provided by RZ is the retrieval of the data component of an object given a complete specification of the content of all its keys. However, RZ can give an application access to the array of key vectors. The application can use any criteria to select one or more objects and then request their retrieval by RZ.

The RZ package was not designed primarily as a component of a database system, and thus contains no special facilities to ensure data integrity in the event of, for example, a system crash.

In summary, RZ was a platform on which database access by name and by time could be built, and which already contained the important feature of segregation of compact and bulky data with memory caching of the compact data. As with all components of the ZEBRA system, RZ was available on all computers used by L3. The remaining requirements listed above had to be provided on top of RZ and led to the creation of the DBL3 database package.

6.3 Features of the DBL3 Package

The functions of the DBL3 package correspond to the requirements outlined in section 6.1. In the present section we describe the principal features of the implementation which have allowed these requirements to be met. Further details can be found in Appendix and in Reference [11].

6.3.1 Database representation in memory

The DBL3 package uses a FORTRAN labelled common block as ZEBRA dynamic store. This can be any of the stores used by the application program. DBL3 operates inside two divisions in that store; a DBL3 system division, of no concern to the user, and the DBL3 data division into which, at the user's request, the data objects are retrieved from the random access medium and kept in memory as long as required by the user.

In the DBL3 data division the package stores data objects as ZEBRA banks or ZEBRA data structures within a main data structure which reproduces, partially and only for the directories in use, the node/key structure of the database files. A node/key structure is maintained in the data division for each database file opened by the user. The skeleton of node and key banks grows up, in response to successive user requests, and stays permanently available in memory (Figure 2b). The data banks, appended to the corresponding key banks, can either be refreshed when the time dependence of their contents requires it, or dropped when they are no longer needed by the user. They can also be marked as temporarily unwanted, so that the system can drop them in case of shortage of memory, at the cost of reaccessing them from the random access device if and when required again.

6.3.2 L3 standard keys and user keys

Objects in RZ directories created by DBL3 contain 7 'standard keys' which are used to implement DBL3 functions not present in RZ. Up to 93 keys are available to the user.

Two standard keys store the period of validity of an object as supplied by the user. One key contains the insertion time. Three other keys contain the indices, pointers and flags needed for data compression. Finally, a key is reserved for a 'source identifier', a user-supplied word indicating the source of the data.

User keys can perform two sometimes overlapping functions. Firstly, keys can be used to identify objects within a directory. For example if all the calibrations for a set of 16 wire chambers were stored in one directory, a user key containing the chamber number would allow retrieval of the calibration for a specific chamber. Secondly, the user keys can contain data to which rapid access is needed. The amount of real time needed to access one word from the key vector of all objects in a directory is typically several orders of magnitude less than the time required to access one word from each disk-resident data object in a directory.

6.3.3 Data compression

DBL3 attempts to minimize the size of the random access file in two ways:

1. For simple data objects in which all elements are of the same type (integer or real), DBL3 packs the data using a default storage precision or a precision specified by the user. The default algorithm for packing searches for the optimum bit-length with which the majority of the data can be packed, the remainder being represented by two data words, the value and the location in the data array. Another packing algorithm, activated on user request, stores only the data elements whose absolute values are greater than a predefined number.
2. Taking advantage of the high probability that a new calibration differs minimally from a previous one, DBL3 looks for a neighbouring object such that the difference with the object to be inserted can be described and packed in the least number of words.

Both forms of compression are invisible to the user and are automatically invoked unless explicitly disabled.

6.3.4 Real time optimization

Many DBL3 directories contain over 10,000 objects and may occupy several megabytes in memory. Retrieving such directories requires seconds of real time, and because of their size, they are likely to be dropped frequently from the RZ directory cache to regain space.

The concept of a partitioned directory has been introduced to attenuate these effects. Subdirectories are automatically created when needed, with keys recording the ranges of validity time and insertion time present in each partition. Normally, only one partition has to be retrieved to satisfy a user request. Partitioning is invisible to the user, who just has to give its characteristics when creating the directory. Subsequent storage and retrieval of data objects in a partitioned directory follow the same rules as for a normal directory, as far as the calling sequences of the user interface subroutines are concerned.

To further optimize real time usage in storing data, DBL3 also provides a facility to store several objects, belonging to the same directory, in one pass. In real time, the gain is about proportional to the number of objects one stores at once.

6.3.5 Journaling

DBL3 journal files are central to its function as a distributed, robust, database system. All updates which affect the database can be recorded in a sequential ZEBRA-FZ file. Examples of updates recorded in the journal file are the addition or deletion of directory trees, the addition or deletion of data objects, the alteration of key values, the deletion of some partitions in a partitioned directory, the definition of aliases and mnemonics in the DICTIONARY directory or the insertion of items in the HELP directory. The

information is recorded in different types of FZ records, consisting of an FZ header, with or without a data part, depending on the kind of action.

DBL3 can read a journal file and apply the corresponding updates to the database. If a system crash damages an RZ database file, all updates which have been journaled since the last backup can be applied to the database. Exactly the same code is used to update copies of the database using journal files sent over networks.

6.3.6 Dictionary

The DBL3 package supports a dictionary in a special directory called **DICTIONARY**. In the dictionary the user can enter, for any of the directories, simple alias names for the pathnames as well as mnemonics for the data object elements (both limited to 8 characters). The **DICTIONARY** directory, which does not make use of the standard L3 keys, is automatically generated, at the top level, when the database is created. For reasons of efficiency, the dictionary is also used to store information about each directory, such as date of last modification, which is stored in the main body of the database.

6.3.7 ASCII data objects and HELP directory

Facilities exist to store and retrieve ASCII files with a record length of up to 80 characters as database objects. This feature is used to store summaries of production job performance in the Job Log database and also to store help information for each database node in a special **HELP** directory.

6.3.8 Interactive Interface

Most of the DBL3 operations mentioned in this report, and some other specific ones, can be controlled interactively. The interactive version of DBL3 is based on the KUIP package [12] and therefore can be used on any usual computer.

6.3.9 Remote Access Facility

The DBL3 package supports access to database files residing on remote computers. This feature was developed for the operation of the L3 attached processor farms, but it can easily be extended to a server/client configuration over high speed links. It is vital in those cases where database information has to be retrieved by programs executing in diskless processors, like for example the 3081/E emulators. This mode of operation requires modifications to four subroutines within the RZ package such that requests for disk input are dispatched to the host control program. In this way, the functionality of both RZ and DBL3, particularly the caching facility, is totally preserved, and the host process is activated only in the case that either a new directory or a data object is needed. Furthermore, to minimize the amount of transactions, requests for retrieval of multiple objects from the same directory are stacked and processed all at once. Entering

data into a database file is achieved through journaling: updates are saved in virtual sequential files which are shipped to the host program for processing.

7 The L3DBSM system

7.1 System Architecture

The basic philosophy of the L3 database management system is that users have free read access to database files but are not allowed to write directly into them. This approach avoids problems of concurrent multi-write access and, at the same time, provides an easy way to transmit updates to other computers.

The client-server scheme is a typical solution to this kind of problem: database updates are sent to a server, checked, executed, and saved in a journal file. Many programs can talk to the server concurrently, but their requests are processed one at a time. An overview of our server system is presented in Figure 1. Clients in the online environment interact with the online server while clients in any other computers update databases through the master server which runs on LEPICS. The master server saves all updates made by online and offline clients in journal files that are distributed to all offsite centers. These journal files are used by the offline servers to update their own databases.

7.2 Online Database Servers

The L3 online system consists of a VAX-cluster of 30 nodes. The main computer is a VAX 8800 which is dedicated to data acquisition. VaxStations and microVaxes are used for monitoring subdetectors and for subdetector local data acquisition.

In the online environment, the server has been implemented as a FORTRAN program running in a dedicated batch queue. There is one server for each database file resulting in a total of eight copies of the same program. Each server is started automatically at boot time, and keeps running until shutdown of the online cluster or explicit stopping procedure. Servers and clients, like all the L3 online programs communicate with each other using a software package called CPC (Cluster Process Communication [13]) written within L3 on top of the VAX/VMS distributed lock manager [14]. Messages are exchanged through *caches* which are memory locations accessible by any process over the cluster.

When started, each server initializes its database, and books two caches that are used to communicate with clients. Servers and clients exchange only messages, not data, to increase communication efficiency. Two additional caches are used for special communications: one is used for general purpose messages (like STOP, PAUSE, CONTINUE), and one to signal Start/End of Run conditions.

The server puts itself in a waiting state, waking up upon two possible events: an interrupt associated with a message in a cache or a VMS event flag. The former case

causes the server to trap in an AST (asynchronous system trap) routine. There the program interprets the message and, if requested, performs operations on the database. The message in the cache contains the operation code (insertion, deletion, purge etc.) and the name of a special file on the disk, where data have been dumped by the client, just before sending the message. This ensures data integrity, because the file is overwritten only after the data have been inserted in the database and after success of the operation has been acknowledged. The latter case is related to all other operations like stopping the server itself, periodic wakeup (see below), pause/continue facility.

Journal files are transferred to LEPICS through INTERLINK [15] by a VMS command procedure spawned periodically by the server. One can set the execution time of this procedure independently for each server and change it at any moment without stopping the server. The command procedure provides for automatic mailing to responsible people in case of network failure or other kinds of errors. An automatic retry procedure handles files that failed to be sent properly to the offline computer.

The number of clients is not predetermined by the system. Client programs must only follow conventions in their naming so that at any time a new client may start to be operational. All server operations, including the auxiliary procedures, are logged to disk files with time stamps. This eases the tracking down of errors or anomalous events.

7.3 Offline Database Servers

In the offline environment the system consists of a master server running on the LEPICS mainframe, and a set of local servers running on each node where the database must be available, i.e. offsite IBMs and the L3 Apollo Domain at CERN, where event scanning and analysis are done.

The master server is a VM/CMS service machine which runs continuously in disconnected mode. The server has two components: an IBM command procedure written in REXX which interacts with the operating system (signal manipulation, file and message exchange, etc.) and a FORTRAN program which uses DBL3 to modify the databases.

Signals from online server and/or other offline programs are processed in the REXX procedure via the IBM IUCVTRAP facility. To speed up operations the two components are always loaded in memory, swapping control when needed. The initialization phase of the FORTRAN program is executed only once at server startup. This phase is actually very time consuming because of ZEBRA and DBL3 initialization.

The master server is normally in a *sleeping* state: it wakes up upon interrupt caused by any of the following conditions:

- A journal file is received in the reader from the online cluster or from a process on LEPICS or on another computer. It is processed and then discarded. At the same time a general offline journal file is updated. Before processing, journal file integrity, database status and disk space availability are checked.
- At predetermined times offline journal files are closed and distributed to offsite computers using the RSCS store-and-forward communications protocol.

- A message from authorized users is received. This allows users to query the server status and to perform control operations from another account without stopping the server itself.

Transactions with clients are asynchronous, *i.e.* acknowledgement is not returned to the client. The master server has similar logging and alarm facilities as the online one. It can accept new clients without any change in the code. The whole system is easily expandable.

The offsite IBM servers are just copies of the master server, with possible modifications due to slight differences in operating system or software installation.

Databases in the L3 Apollo Domain are updated by a FORTRAN program which runs in the background on an Apollo DN10000 computer. The exchange of files and messages between this program and the LEPICS server uses TCP/IP and is carried out by two small automatic procedures (one in each computer) that ensure the transfer and data integrity even in case of network failure. The Apollo server is seen by the master server as another IBM server.

Clients running on offline computers must send their updates to the master server on LEPICS. If the client resides on an IBM computer RSCS is used; if it resides on the Apollo Domain the updates are transferred to a specific area from where a dedicated procedure sends them to LEPICS.

8 Manpower, Performance, Problems, and Improvements

8.1 Manpower

Writing the DBL3 package required between two and three man years of effort. Maintenance of the package absorbs about one quarter of one person's time. The additional code which makes up the L3DBSM system also represents between two and three man years of work. Maintenance of the L3DBSM servers together with the management and recovery of database files is a half time job spread over a small number of experts who must be available at very short notice when problems arise.

The design of the individual subdetector database structures, and the creation of subdetector-specific interfaces for both users and programs, required typically one man-year per subdetector. Maintenance of the subdetector-specific databases absorbs a small fraction of one person for each subdetector.

8.2 Performance

For most users the relevant performance figure is the elapsed time required to complete an operation. The speed of most DBL3 operations is limited by the latency of disk

access rather than by processor speed, so the CPU time consumption is normally far smaller than the elapsed time.

8.2.1 Insertion

All insertion is performed by servers, so the speed at which a server can process insertion requests can become critical. Normally, the server program opens the database RZ file when an insertion request arrives and closes it immediately after the insertion. The processing rate is then dominated by the opening of the database and is typically 3 insertions/s on the LEPICS IBM 3090 (VM/CMS) system and 10 insertions/s on the VAX 8800 (VMS) system. These rates can be increased by more than one order of magnitude by keeping the database opened between updates.

8.2.2 Retrieval

Retrieval times can vary widely depending on the size of the objects retrieved, on whether they are close to each other or dispersed within the RZ file, on the speed of the disk hardware and on the effectiveness of any disk caching.

In a typical run of the L3 reconstruction program using the full 1990 database on the LEPICS IBM 3090, the initialization phase, which is dominated by database access, took 73 s. In this time the program retrieved 2002 objects from 107 directories. The total volume of the data objects was 2.7 Mbytes. This performance is consistent with the 17 to 31 millisecond access time of the IBM 3380 disks holding the database. The test was performed on a lightly loaded machine. Under heavy load, contention for disk access reduces this speed, but disk caching in memory (introduced on LEPICS in early 1991) may reduce the load dependence.

One of the objects retrieved during reconstruction initialization is the data structure describing the detector geometry. This single object has a size of 569 kbytes and takes 2.5 s of elapsed time and 150 milliseconds of CPU time to retrieve and unpack as a ZEBRA data structure. If the same object has recently been retrieved by another user, the disk caching reduces the elapsed time for the second access to 150 milliseconds.

At the other end of the scale, retrieval of many small objects which are contiguous on disk takes about 6 milliseconds per object on the IBM.

On an Apollo DN10000, which has a CPU equivalent to half an IBM 3090, the access times are typically 2 to 3 times longer than on the IBM.

8.3 Problems

8.3.1 RZ package

It should be pointed out that RZ was not designed to be a database package or even to provide an access method meeting the high reliability demands of a database package.

The most serious problem has been the possible corruption of the database file after a crash of the server. Although not frequent (five crashes on the VAX cluster and one

on LEPICS in 1990), the corruption of the database file is critical, online, to the data acquisition, and offline, to the data processing.

In the online environment, only processes non-essential to the data acquisition use the database directly, like data and detector monitoring processes. Other processes, *e.g.* initialization processes, access their data from dedicated files produced at earlier times from the database. In any case, the database must be quickly rebuilt (typically within an hour). This is particularly true online where a lack of monitoring could result in a loss of physics data, in addition to the loss of data that are produced by the detector monitoring systems. The lack, in RZ, of adequate tools that could allow a quick localisation of the area of corruption does not facilitate the rebuilding of the database. Another consequence of the corruption of the database was the reluctance of application programmers to use some elegant features of the DBL3 package, like the "store-the-difference" form of data compression, that would prevent the database from being rebuilt if one journal file were lost. In addition to the database files, the journal files can also be corrupted. This can lead to differences between the online and offline databases requiring lengthy manual interventions.

Another limitation, inherent in the use of direct access file, is the fact the the size of the database file does not decrease immediately after a purge but only after two changes of format (RZ to FZ and FZ to RZ).

8.3.2 DBL3 package

Although the existing interactive interface is considered tolerable by database experts, a weak point of the DBL3/RZ database system is the lack of a user-friendly interactive interface. Interfaces that protect the user from the DBL3/RZ package technicalities, check the integrity of the input data, and present it in a friendly way are particularly needed in the online environment where the persons on shift, non DBL3/ZEBRA specialists, must consult and update the database frequently. Each subdetector group has developed its own interface separately. The need for a standard user-friendly interface that could run on VAX, IBM and Apollo computers soon became evident. Such an interface could be implemented in a relatively short time.

There are many possibilities for unintentional misuse of DBL3 which produce programs which work correctly but take many orders of magnitude longer than necessary. The following two examples of these possibilities reflect real experiences in L3 which wasted resources until the problems in database structure and application code were understood and corrected.

1. It is very important that information which will be retrieved frequently, and is distributed over many objects in a directory, is stored in the 'keys'. All the keys in a directory can be retrieved in one or a few disk accesses. Conversely, retrieval of one word from each of 1000 data objects normally requires at least 1000 disk accesses.
2. When accessing data from many directories, programs often have insufficient space in the ZEBRA system division to cache all the directories which are in use. To

prevent a directory being read from disk for each data access it becomes very desirable to perform all retrievals from one directory before moving on to the next.

8.3.3 Operating systems

On IBM with VM/CMS operating system, the size of the database file seen by an application program remains constant, unless the minidisk that contains the database is reaccessed. If the server expands the database file the application program may see a corrupted database and crash.

On Apollo, the Aegis operating system does not allow concurrent write and read. Thus, the update of the database can only happen after the release of the database file by all applications. This problem can be circumvented using the UNIX operating system.

8.3.4 Networks

A few journal files were lost or corrupted during transfer through the various networks. We counted five errors in about 20000 transfers. The consequences are benign because journal files are kept on the online computer and on LEPICS for several weeks and can always be retransferred.

8.4 Possible Improvements

The functions listed in this section are not only desirable, but likely to be implemented.

8.4.1 Read Server

In the current L3DBSM system servers are used only to insert data in the database; the complexity and inefficiency of read access through servers has thus been avoided. However, most of the problems associated with operating system peculiarities could be solved by using a read server system. In addition, when many processes are attempting to access the same database objects, a read server could reduce the delays. With its heavy use of caching (in the DBL3 node/key structure and the ZEBRA system division) the DBL3 code is already close in its functions to those of a read server.

8.4.2 Partial Database Support

As our database files grew larger, the interest for partial databases on offsite centers became stronger. The maintenance of a partial database requires either the creation of customized journal files by the LEPICS server or customized servers on offsite centers.

9 Conclusion

We have described the technical details, strengths and weaknesses of the DBL3 package and the L3DBSM system. We believe that the concepts of L3DBSM/DBL3 were appropriate for the L3 experiment and will also be appropriate for large experiments now being planned.

The growth of calibration needs and book-keeping needs as the LHC/SSC era approaches is likely to be more than matched by the rising capabilities of computing hardware. Since L3DBSM/DBL3 has worked well for L3, it would thus be feasible to offer L3DBSM/DBL3 unchanged as the solution for the next generation of experiments. However, a better approach would be to examine how the known weaknesses of the system could be overcome, using improved HEP software or commercial products, while preserving the basic concepts and services of L3DBSM/DBL3.

Enquiries about the availability of the DBL3 Package and the L3DBSM servers should be addressed to R. Mount at CERN (MOUNT@LEPICS.BITNET). The CERN program library is a prerequisite for the DBL3 package.

Appendix: The DBL3 User Interface

In this section we list, with brief explanation, the main user-callable subroutines in the DBL3 package.

Initialization and Termination

DBINIT: initialize the memory resident structures and retrieval of the dictionary for each random access file. The files themselves must be opened by FORTRAN OPEN statements before DBINIT.

DBEND: close all database files.

Creation of Directories

DBCRSD: create a directory with the seven 'standard keys' and up to 93 user keys. Directories which may become very large may be flagged for automatic partitioning.

Storage and Replacement of Data

DBENTR: store a ZEBRA bank or data structure together with any user keys. The validity period of the data must be specified. Data compression and the creation, if necessary of a new directory partition, are automatic. The keys and data can be stored in the memory resident node/key structure for later access within the same program.

If an existing object should be replaced (by a better calibration for example) DBENTR is used to insert a new object with the same validity period and the same user keys. The later insertion time of the new object renders the old one normally invisible.

DBREPL: replace an old object by a new one with different user keys. As with DBENTR the old object is logically superseded by the new one and is not physically deleted.

DBABWR: insert an ASCII file as a database object.

Retrieval of Data

DBUSE: the principal data access routine of DBL3. Objects are requested by specifying the pathname, the validity time (typically the data acquisition time of the event being processed) and any required restrictions on the user key values.

If the required object is already in memory attached to the node/key structure, DBUSE returns almost immediately. Otherwise, any necessary additions to the node/key structure are created and the data are retrieved from mass storage and attached to the structure. DBUSE returns the addresses of the keys and the data.

DBFREE: mark data object(s) attached to the node/key structure as available for deletion if the DBL3 data division becomes full.

DBEFOR: ignore all objects inserted after a given time. This implies that logically deleted objects may become visible again.

DBABRD: retrieve an ASCII file from the database.

Other calls are available for users who prefer to insert and retrieve FORTRAN vectors rather than ZEBRA banks. Calls are also available to delete objects or directories when logical deletion alone would not be appropriate.

References

- [1] *The Construction of the L3 Experiment.*
L3 Collaboration, Nucl. Instr. Meth. A289, 35 (1990)
- [2] *ZEBRA user guide.*
R. Brun, J. Zoll, CERN Program Library, (1987) (unpublished).
- [3] *Database System for HEP Experiments.*
R.P. Mount, Computer Physics Communications 45, 299 (1987).
- [4] *Requirements for the L3 Database.*
L3 Offline Software Group, L3 report no. 315, (1984) (unpublished).
- [5] *A KAPACK based L3 Data Base Management System.*
A. Gurtu, L3 report no. 370, (1985) (unpublished).
- [6] *Modular Architecture for Online System.*
A. Vascotto, Data Handling Division, CERN, DD/86/17, (1986) (unpublished).
- [7] *The L3 Online Window Manager Facility.*
T. Wenaus, M.I.T., L3 Note, (1989) (unpublished).
- [8] *OS9/68000 Operating System Users's manual and Technical Manual.*
Microware System Corporation, (1987).
- [9] *ZEBRA Reference Manual, FZ Sequential I/O.*
J. Zoll, CERN Program Library, (1989) (unpublished).
- [10] *ZEBRA Reference Manual: MZ Memory Management.*
J. Zoll, CERN Program Library, (1988) (unpublished).
- [11] *DBL3, A Software Utility Package for the L3 Database.*
S. Banerjee et al. , L3 report no. 701, (1989) (unpublished).
- [12] *KUIP - Kit for an User Interface Package.*
R. Brun and P. Zanarini, Data Handling Division, CERN, CERN Computer Center Program Library, Long Write-up I202 (1988) (unpublished).
- [13] *The L3 Cluster Process Communication Facility.*
T. Wenaus, M.I.T., L3 Note, (1989) (unpublished).
- [14] *Guide to VMS Programming Resources*
Digital Equipment Corporation, (1988).
- [15] A product of Interlink Computer Sciences, Fremont, California (USA).

**Table 1. Space used by subdetector databases in Mbytes
(1990 running period)**

<i>Type of Data</i>	<i>Vertex Det.</i>	<i>Electromagnetic Cal.</i>	<i>Hadron Cal.</i>	<i>Muon Chambers</i>
<i>Fixed Data</i>	0.04	2.3	0.5	0.07
<i>Slowly Varying Data</i>	1.8	1.2	0.8	0.6
<i>Periodic Calibrations</i>	5.6	50.3	47.0	7.2
<i>Continuously Monitored Parameters</i>	69.6	0.4	16.6	40.4

Captions

Figure 1: Overview of the L3DBSM system. The online system consists of a VAX-cluster of 30 nodes performing data acquisition and monitoring. The central offline computer, LEPICS, is an IBM 3090/180E. All computers at offsite centers are IBM 3090. The L3 Apollo Domain at CERN, where event scanning and analysis are done, is considered by the L3DBSM system as another offsite center.

Figure 2a: Part of the muon chamber database structure.

Figure 2b: Example of a corresponding node/key structure present in memory during database use.

Figure 3: Retrieval of data objects.

Figure 1

Figure 2a

Figure 2b

Figure 3