Data Handling in KLOE

I. Sfiligoi,
INFN LNF, Frascati, Italy,
for the KLOE collaboration[1]

1) Introduction

KLOE[i] is the experiment for which the INFN DAΦNE φ–factory in Frascati, Italy, was built. Its main goal is the measurement of CP violation at sensitivities of $O(10^{-4})$, but it is also capable of investigating a whole range of other physics. The most interesting studies include kaon form factors, kaon rare decay and radiative φ decays measurements.

KLOE is designed to acquire $\sim 10^{11}$ events per year at the full DAΦNE luminosity of $5x10^{32}$ $cm^{-2}s^{-1}$. The total number of acquired events will be of the order of 200 billion.

In addition to the physics event data, there are also several other kinds of data that must be stored and manipulated; calibration, slow control and run condition data are also very important for reaching the desired physics results.

Taking into account all the data sources, the estimated space requirement for KLOE is ~1 PB. Obviously, this large amount of data requires an efficient, reliable and easy to use system.

Given the required computing power and the mass storage requirements, the best solution would be a supercomputer–like system, but the financial effort for this kind of system would be too high. Moreover, the most advanced features of such a system, such as common addressing space and automatic hardware

1 **The KLOE collaboration:** M. Adinolfi, A. Aloisio, F. Ambrosino, A. Andryakov, A. Antonelli, M. Antonelli, F. Anulli, C. Bacci, A. Bankamp, G. Barbiellini, G. Bencivenni, S. Bertolucci, C. Bini, C. Bloise, V. Bocci, F. Bossi, P. Branchini, S. A. Bulychjov, G. Cabibbo, A. Calcaterra, R. Caloi, P. Campana, G. Capon, G. Carboni, A. Cardini, M. Casarsa, G. Cataldi, F. Ceradini, F. Cervelli, F. Cevenini, G. Chiefari, P. Ciambrone, S. Conetti, S. Conticelli, E. De Lucia, G. De Robertis, R. De Sangro, P. De Simone, G. De Zorzi, S. Dell'Agnello, A. Denig, A. Di Domenico, S. Di Falco, A. Doria, E. Drago, V. Elia, O. Erriquez, A. Farilla, G. Felici, A. Ferrari, M. L. Ferrer, G. Finocchiaro, C. Forti, A. Franceschi, P. Franzini, M. L. Gao, C. Gatti, P. Gauzzi, S. Giovannella, V. Golovatyuk, E. Gorini, F. Grancagnolo, W. Grandegger, E. Graziani, P. Guarnaccia, U. v. Hagel, H. G. Han, S. W. Han, X. Huang, M. Incagli, L. Ingrosso, Y. Y. Jiang, W. Kim, W. Kluge, V. Kulikov, F. Lacava, G. Lanfranchi, J. Lee–Franzini, T. Lomtadze, C. Luisi, C. S. Mao, M. Martemianov, M. Matsyuk, W. Mei, L. Merola, R. Messi, S. Miscetti, A. Moalem, S. Moccia, M. Moulson, S. Mueller, F. Murtas, M. Napolitano, A. Nedosekin, M. Panareo, L. Pacciani, P. Pagès, M. Palutan, L. Paoluzi, E. Pasqualucci, L. Passalacqua, M. Passaseo, A. Passeri, V. Patera, E. Petrolo, G. Petrucci, D. Picca, M. Piccolo, G. Pirozzi, C. Pistillo, M. Pollack, L. Pontecorvo, M. Primavera, F. Ruggieri, P. Santangelo, E. Santovetti, G. Saracino, R. D. Schamberger, C. Schwick, B. Sciascia, A. Sciubba, F. Scuri, I. Sfiligoi, J. Shan, T. Spadaro, S. Spagnolo, E. Spiriti, C. Stanescu, G. L. Tong, L. Tortora, E. Valente, P. Valente, B. Valeriani, G. Venanzoni, S. Veneziano, Y. Wu, Y. G. Xie, P. P. Zhao, Y. Zhou

recovery, are not essential for the KLOE computing environment, although they would significantly simplify the system management.

The solution adopted at KLOE is based on a set of medium–sized servers. Moreover, in order to better follow the changing market trends, all the KLOE software is developed for several platforms.

2) KLOE software environment

As in many HEP experiments, the KLOE software environment can be divided in three quite distinct classes:

- online or data acquisition (DAQ) software
- offline or data analysis software
- montecarlo event generators

In this section the description of the parts specific to each environment class can be found, while the more general parts of the system are described in separate sections later on in the document.

2.1) The KLOE data acquisition

The KLOE DAQ[ii] can be divided in two logical blocks;

- The *low level block* is made of the Front–End electronics, ADC and TDC VME boards and a complex readout system implemented using custom devices. This logical block is implemented using custom hardware and will not be mentioned anymore in this paper.
- The *high level block* is made of VME CPU boards, the online farm of computing servers and the interconnecting switched network. Only general purpose commercial hardware is used in this logical block.

The main task of the high level block is to read the pieces of events from the low level block, merge them into single event frames and save them for offline processing. Since the number and size of events are very demanding for any existing database system, the event data are stored as YBOS[iii] files of reasonable size (~500 MB at the moment) and only a reference to them is stored in the database.

All the other data treated by the DAQ, such as configuration, environment and monitoring data, are much smaller and are thus stored directly in the database.

The software used in the DAQ environment is mostly in–house developed and is written in ANSI C. However most of the low level activities, such as network communication, are handled by commercial software.

2.2) The KLOE offline environment

The principal task of the offline programs is the reconstruction and the analysis of the acquired event data. The reconstruction uses in addition also several different types of related information, such as the detector geometry and calibration parameters.

The offline software can be divided in four distinct categories that access data in different ways:

Software category	*Description*
quasi–online reconstruction and calibration	programs that run as part of the production process and access the recently acquired data
reconstruction reprocessing	programs that run as part of the production process and reprocess the data acquired in the past
official data analysis	programs that run as part of the production process and do physics analysis on the reconstructed data
user programs	all other programs

The offline software environment is based on a FORTRAN framework, namely Analysis_Control[iv] (A_C), where software modules can be combined to obtain the desired result. Actually, most of the modules are common to all the categories of programs. Also the input and output are managed via specialized modules, so that different storage policies can be easily implemented by simply plugging in different IO modules.

2.3) Montecarlo event generators

The montecarlo event generators are a very special kind of program; their task is to simulate physics events and produce raw event data that mimics the DAQ output as much as possible, while maintaining as much information as possible for the understanding of the problems to be studied. For this reason, the output is in the form of YBOS files, where additional, montecarlo specific banks contain the event description.

The montecarlo program, called GEANFI[v], is based on the GEANT[vi] package, version 3.

3) The KLOE database system

The KLOE database system (DBMS) is made of two components:

- the *offline database*, used mostly by the offline and montecarlo programs
- the *online database*, used mostly by the DAQ

3.1) The offline database

The offline database needs to manage large data entities, that change slowly in time. Thus the database system must be optimized for information retrieval and ease of use.

The system of choice is based on the ***HepDB***[vii] package, which has the advantage of being easy to integrate inside the A_C–based offline environment. It is also well suited for the management of large sized entries.

The data managed by the offline database are:

- geometry data
- calibration parameters
- run conditions

These data are accessed via an in–house developed FORTRAN library, that acts as a front–end to the ***HepDB*** FORTRAN library calls. This additional level simplifies the access to the database, grouping the general ***HepDB*** calls to more specific routines.

The data insertion is indirect; to handle concurrent updates and to prevent accidental overwriting of the stored data, the user is asked to prepare the data in a file and submit it to an update–specific program that makes the actual update.

The database repository resides on an NFS exported file system, allowing access to the database data to all the KLOE computing nodes.

3.2) The online database

The data managed by the online database are instead quite small and well structured and also get updated quite often. Moreover, the access policy to these data can vary widely across different applications. As a consequence, the online database must be very efficient and flexible under any condition.

The system of choice is based on the relational database system (RDBMS) paradigm. This kind of systems have proven to have all the desired qualities and there are several commercial RDBMS available and most of them run on several of the KLOE supported platforms. Last but not least, this kind of systems are also quite programming language independent.

The data managed by the online database can be classified in the following way:

	access rate	*accessed data*	*update rate*	*update type*
DAQ configuration	once per run	last entry	once every few runs	append
	several times per hour	any*		
run information	several times per minute	any	once per run	append
slow control data	several times per minute	last entry	once every few seconds	append
	a couple of times per day	any		
event data bookkeeping	very often	any	very often	insert update delete
support data (such as identifier descriptions)	very often	any	rarely	insert update delete

* only a small subset of the data is typically in use at any given time

The above table shows how the access and update policies of distinct classes are very different and, even inside the same class, there can be different access policies for different kinds of data. Accurate database planning was therefore essential for obtaining acceptable response times.

The database system is composed of an in–house developed non–privileged daemon (***database daemon***), a user library and a commercial RDBMS. The users of the database system, however, are only aware of the user library.

The ***database daemon*** and the commercial RDBMS run on a single, trusted central server. All communication between the two components is local and is based on the commercial product specific interface.

On the other hand, applications can be run on any network–connected machine. The data exchange between the applications and the database system is based on a client/server mechanism between the user library and the ***database daemon***. Using TCP/IP sockets, the requests are sent to the ***database daemon*** which processes them, using the commercial RDBMS if needed, and replies on the same socket connection.

This approach was chosen for several reasons;

- the dependence on a specific commercial RDBMS is minimized, without however sacrificing the system performance
- special configuration of the client side is not needed

- part of the advanced logic is stored in the daemons, allowing:
  - additional checks and restrictions
  - fast central caches (for example, the DAQ configuration cache reduces the typical access time to as little as 3%)
- since most of the access and update logic is located inside the daemons, most of the updates to the database structure do not require any modification of the final applications

The commercial RDBMS of choice is the IBM DB2 Universal Database[viii]. However, any other SQL–based RDBMS can be easily used with slight modifications to the ***database daemon***.

The online database system can work also without an underlying RDBMS, although with some limitations. A simplified version of the ***database daemon*** has been implemented to allow the users to work in stand alone environments such as home PCs. Only a subset of the normally answered requests are served by this daemon, but the implemented ones are enough for normal operation. The required data are read from a file that the user has obtained from the central database system. Only the data the user is interested in are transferred in order to make the file as small as possible.

4) The KLOE archiving system

Until this point, no clue has been given as to where the data managed by KLOE actually resides. Since this data will be used for a long time, it's obvious that it must be stored on permanent media, but the choice of which media to use may not be so trivial.

At the time of writing, the fastest and most reliable way of storing and accessing the data is based on hard disk arrays. However, the event data managed by the experiment is estimated to reach the petabyte range, but no existing disk–based system is able to support such amounts of data.

As of now, the only available systems that can easily scale to the required size are based on tape libraries. Accessing the data in a tape library can be however quite inefficient, since the access type is essentially sequential.

The solution adopted at KLOE is a mix of the two worlds. Tape libraries are used to store the event data, while a set of arrays of disks of reasonable size (a total of ~1 TB at the moment) is used for database–managed data and as a cache for the data on tape. In this section, the tape storage and cache mechanism is described.

The archiving system is composed of a set of in–house developed non–privileged daemons (***archiving daemons***), a user library and a tape library manager.

The ***archiving daemons*** themselves can be further divided in:

	Components	***Description***	***Location***
storage managers	***archADSM***	Interface to the tape library manager. Their tasks include content listing and file archival and retrieval.	one on each tape server
disk space managers	***filekeeper*** ***spacekeeper***	Manage the files on the disk pools. All the files are created and deleted by these daemons.	one for each disk pool
archival director	***archiver***	Coordinates the file movement from the disk pools to the tape library.	one instance on a dedicated server
cache manage	***retrieve***	Manages the user requests for event data files and coordinates the file movement from the tape library to the disk pools.	one instance on a dedicated server

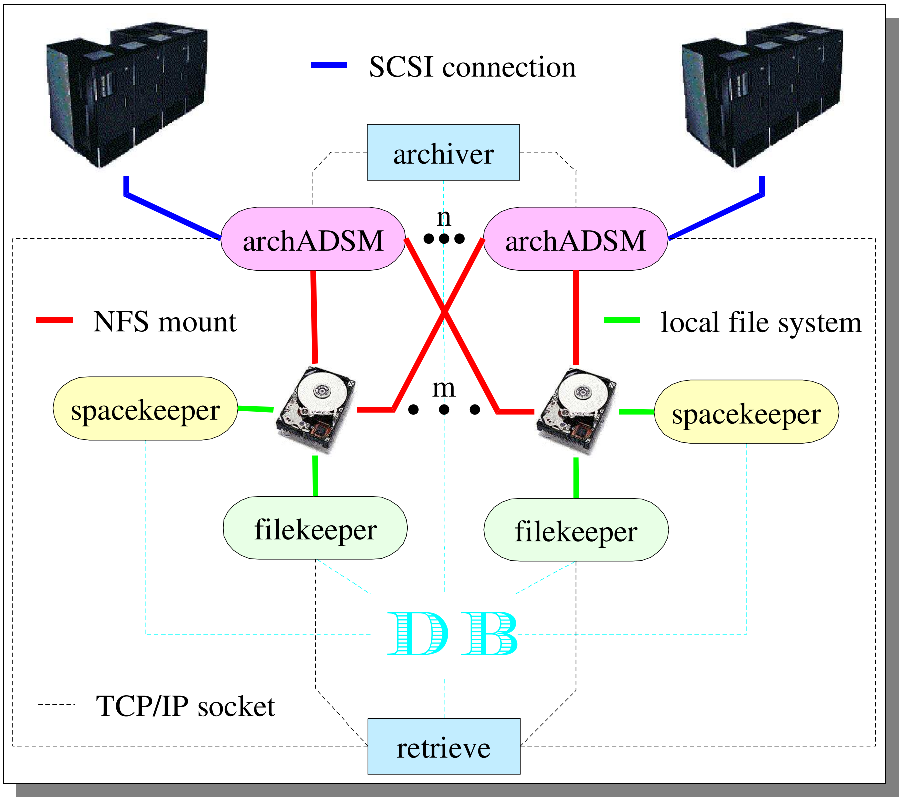

Figure 1: A schematic view of the archiving system

The different pieces of the archival system communicate by means of TCP/IP sockets. All the archiving daemons but the archival director have a standard communication interface, and the related programming library, that is used to ask for services. Moreover, some database data are used to synchronize their work.

The archiving, i.e. the file storage on the tape library, is fully automated and is managed by the archival director using the following scheme:

- periodically look for files that are ready to be archived
- if such files exist,
  - give the list to one of the storage managers
    - ➡ the storage manager actually moves the files to tape
  - in case of errors retry with another storage manager
  - update the status of the archived files

Once the files are archived, the disk space used by them can be freed. This task is carried out by the disk space managers of the related disk pools. Each of them monitors the disk pool usage and when space is needed for new files, some of the archived files get deleted.

The caching mechanism instead is user driven. When a file (or a set of files) is needed, the application, using the supplied user library, requests the cache manager to put the necessary files onto the disk pools. The following scheme is used by the cache manager:

- find out which of the requested files are not already on a disk pool
- for each such file,
  - allocate the necessary space (via a disk storage manager)
  - ask the associated archival manager to copy the data from tape to the specified disk pool

Note that the storage managers are the only connection between the tape library manager and the other parts of the archiving system. In the current implementation, based on the ***IBM ADSM***[ix] product, each storage manager is responsible for the data stored in its logical tape library. This approach allows the use of multiple storage servers even with only one physical tape library, in order to scale easily with the increasing throughput and to prevent single points of failure. Although this system does not guarantee 100% uptime (the data on a server that is down are temporarily unavailable), most of the archiving system can be operational 99.9% of the time.

The choice of a specific tape library manager for the current implementation does not commit us however to that specific product. The external interface of the storage manager daemons is very general, allowing the creation of other similar daemons, specific to other products, that can coexist or substitute the current ones. This gives the experiment the flexibility to use the best tape library manager available at any given time.

5) The KLOE Integrated Dataflow (KID)

As stated above, the access to the files is not integrated at the system level, but requires an additional layer to work. Moreover, the DAQ allows access to the event data even before they are written to disk and some programs analyze also this type of data.

To allow uniform access to all the sources of event data, the KID system has been developed. It is composed of a user library, an A_C input module, and a non−privileged daemon. The later is needed to give access to the resources located on other computing nodes; one copy of it must run on every server that wants to export its resources. TCP/IP sockets are used to communicate with rest of the KID system.

The user interface to the KID system is based on URIs[x] (Uniform Resource Identifiers) i.e. simple text descriptors that are easy to understand and remember. The KID URIs are modular, supporting several types of access, including the recursive URIs.

In the following table the currently supported URIs are listed:

<table>
<tr><th>Description</th><th>URI</th></tr>
<tr><td>simple file access</td><td>YBOS:filename[?options]</td></tr>
<tr><td>memory data access</td><td>SPY:buffername[?options]</td></tr>
<tr><td>remote resource</td><td>REMOTE:URI@host[:portnr][?options]</td></tr>
<tr><td>merging of several sources</td><td>MERGE:<<i>URI</i>|<i>URI</i>|...|<i>URI</i>>[?options]</td></tr>
<tr><td>database driven resource selection, relative to different kinds of data</td><td>RAW:SQL where clause[?options]
DATAREC:SQL where clause[?options]
MC:SQL where clause[?options]</td></tr>
</table>

The advantage of this method is ease of use. The programmer does not need to parse the user input to select the correct data source, the KID package does it automatically. Moreover since the URI syntax is defined only in one package, different applications can share the same user interface.

Obviously there are also some drawbacks. The most significant is the impossibility to check at compile time the correctness of a static URI, delaying some of the otherwise static checks to run time. A solution to this problem could be in the precompiling of the URIs, but the problem does not justify the effort of implementing the precompiler.

Let us now examine a couple of examples; the following images represent the timetable of a couple of requests to KID.

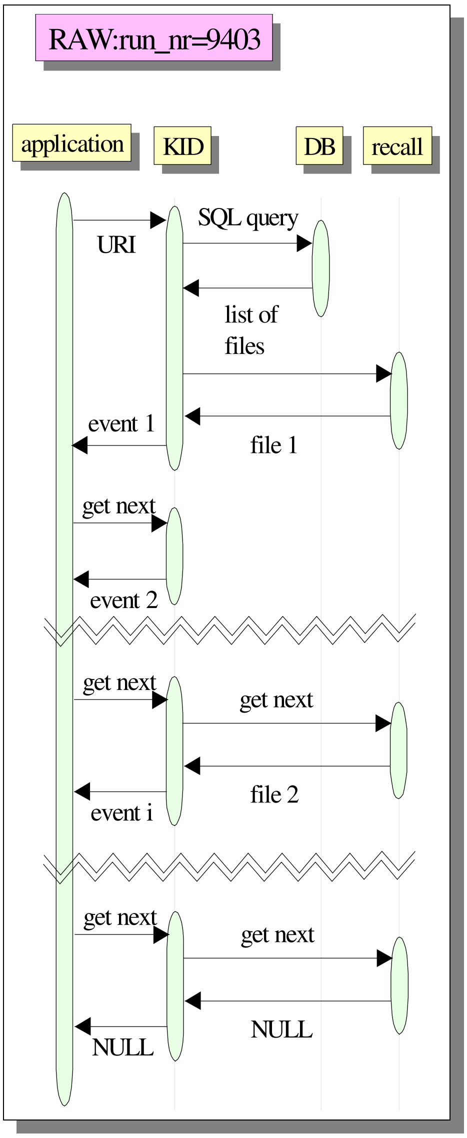

Figure 2: KID timetable, example 1

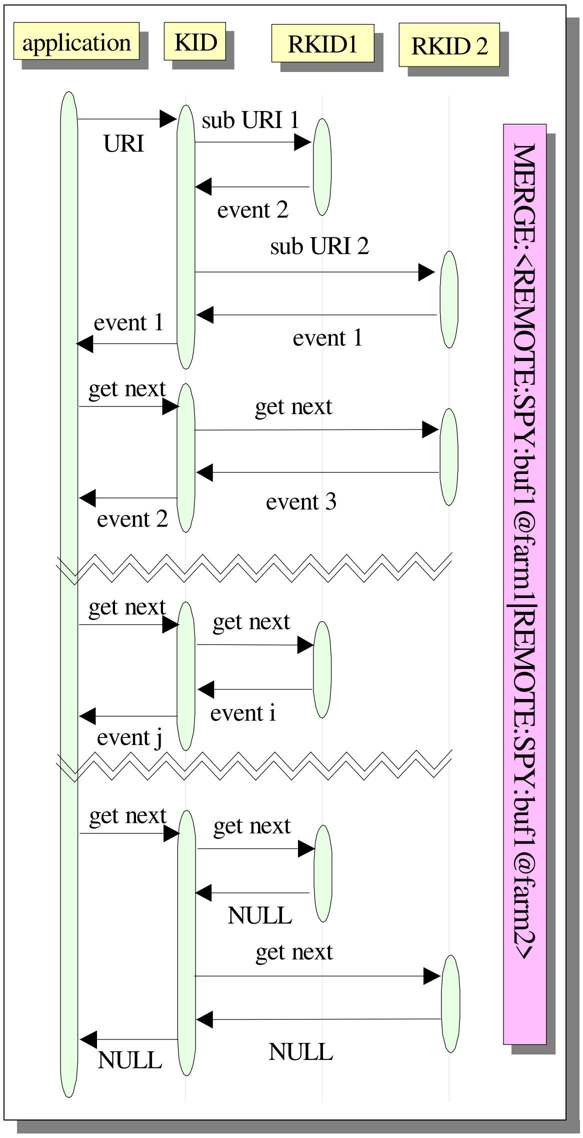

Figure 3: KID timetable, example 2

The left figure shows the response of KID to a request for a complete set of events acquired during a specific run; note that no other information is given. The KID package manages to identify the files related to that run, retrieves them from the tape library if needed, unpacks them and supplies the calling application with the requested events.

The right figure instead represents the KID's response to a request for events from two memory buffers, located on two different servers. Also in this case, the KID package does everything by itself, making the necessary connections and sorting the events in ascending order, based on the event number.

6) Security considerations

Although the scientific environment is quite an open one, some security measures are essential. In KLOE, three types of protections are in use:

- TCP/IP filtering

  prevents outside nodes to access unprotected services
- IP node authorization lists

  dedicated KLOE machines are allowed to connect to additional services
- user authorization lists

  KLOE users on general purpose machines are given an account specific 128–bit key that grants access to additional services

Apart from the TCP/IP filtering, implemented in the router itself, the other security tasks are managed by the in–house developed non–privileged authorization daemon. This daemon, executing on a trusted central server, holds the list of the authorized users/IP nodes for the single services and communicates with the service providers themselves via standard TCP/IP sockets.

When a service provider receives a request from a user/IP node, it asks the requester to encrypt a unique 128–bit message (created for example based on the current time) using the ***Rijndael***[xi] block cipher. The clear and encrypted message are then sent to the authorization daemon for authentication.

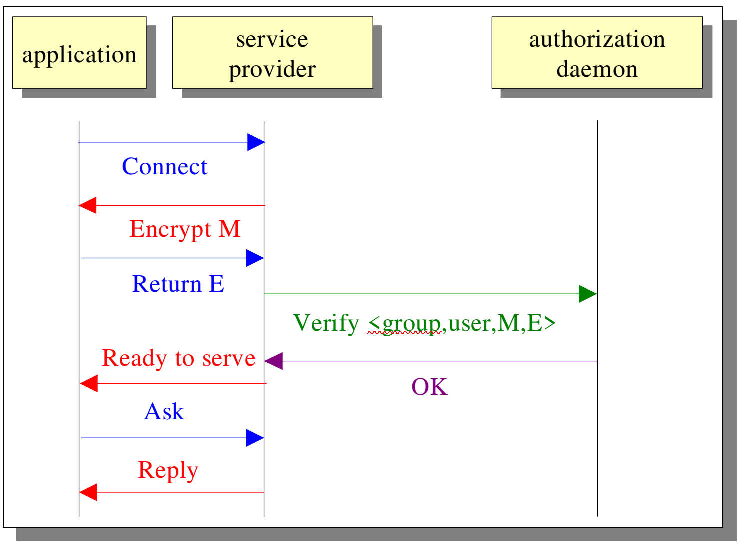

Figure 4: Authentication timetable

The security measures are applied only at the level of the first connection. Once an application has gained access to a service, the data are sent across the network in clear. However, in case of very sensitive data that need to be managed by the KLOE data handling system, the full 128–bit ***Rijndael*** encryption library is available.

7) The KLOE computing environment overview

As described in the introduction, the KLOE computing environment is based on a set of medium sized servers, connected via a switched network. The currently managed system runs several operating systems (IBM AIX, Sun Solaris, HP–UX, Compaq Tru64 UNIX and Linux) and all the KLOE software is written to work at least on these platforms.

The KLOE servers can be classified in five distinct categories:

- DAQ servers
- offline number crunching servers
- disk servers
- archive servers
- the database server

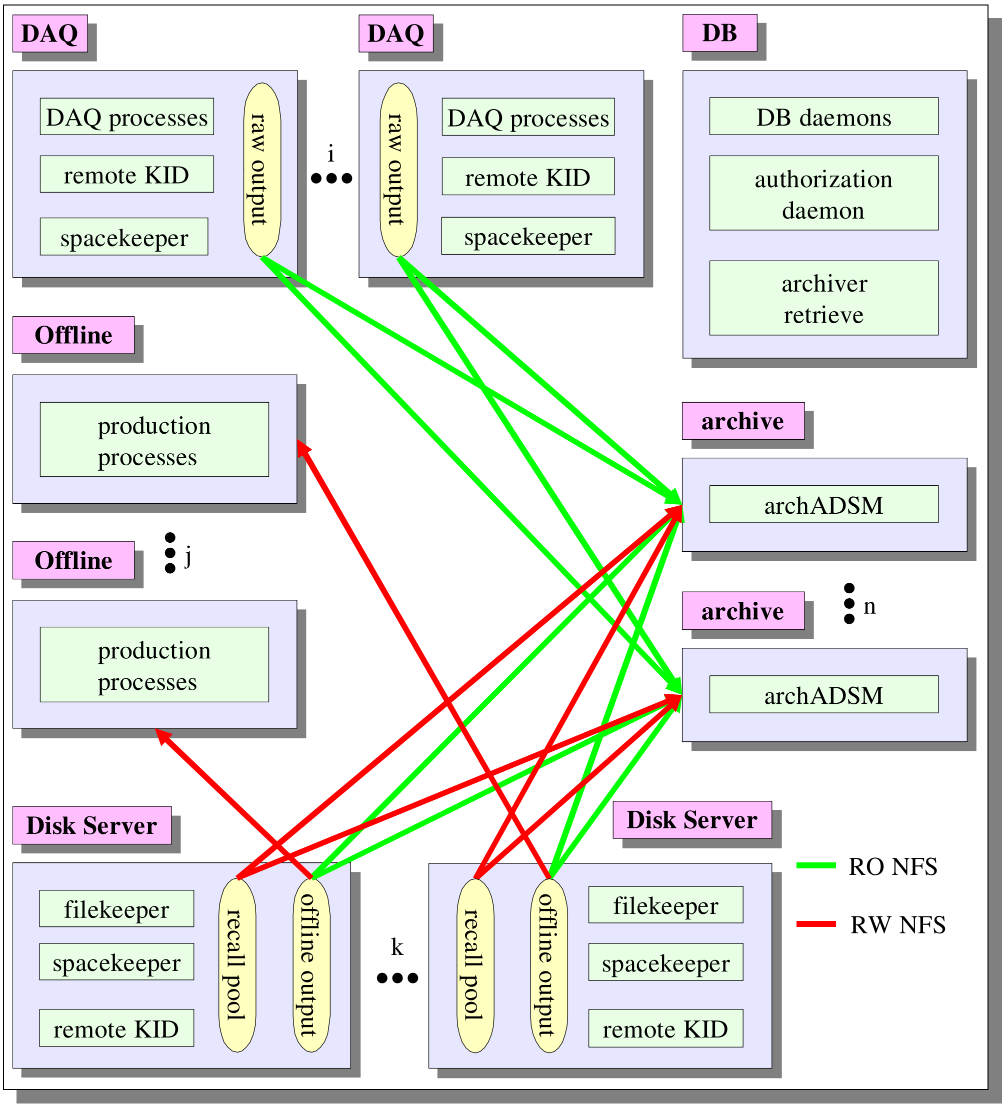

Figure 5: A schematic view of the current KLOE computing environment

All of them are connected using a high speed switched network; the DAQ and offline servers use Fast Ethernet connections to the switch, while the disk, archive and database servers use Gigabit Ethernet connections. The fast network is needed since most of the data need to be moved from a server to another. For example, event data acquired on the DAQ nodes must be moved to the archive servers for archiving, the output of the offline production is first stored on the disk servers and then moved to the archive server for archiving, and so on.

The communication between nodes is fully TCP/IP based; remote file systems are mounted using NFS over TCP/IP, while the KLOE specific services, like database and archiving, are based on TCP/IP sockets.

8) Conclusions

The KLOE data handling system is designed to support a sustained DAQ throughput of 50 MB/s at full DAΦNE luminosity for a total storage requirement of ~1 PB.

The whole system has been tested with simulated traffic well beyond the requirements and until now we haven't found substantial problems in reaching the desired throughput in such conditions. Another series of tests were made to check the actual scalability of the archiving system. Unfortunately we are unable to test the system with a petabyte of data, but we have done some tests using the same amount of files that is expected at the end of the KLOE experiment. Also in this case, no problems have been found.

Aside from tests, we have also gone through the first period of physics data acquisition. This running period was much less demanding than the next runs will be, having a sustained DAQ throughput of only ~3 MB/s and a total storage requirement of ~4 TB. However, it was a good test for the whole system. Some minor problems, especially regarding system monitoring, were discovered and fixed, but otherwise the system proved to be reliable enough for the production environment.

References:

i **KLOE, A General Purpose Detector for DAΦNE**,
The KLOE Collaboration,
LNF−92/019 (IR), April 1, 1992

ii **The KLOE Data Acquisition System,**
The KLOE Collaboration,
LNF−95/014 (IR), March 29, 1995

iii **YBOS, Programmers Reference Manual,**
CDF Computing Group
CDF Note No. 156

iv **A Beginner's Guide To Analysis_Control And Build_Job,**
M. Shapiro, E. Sexton−Kennedy, D. R. Quarrie,
CDF Note No. 384

v **The KLOE Montecarlo GEANFI**
A. Antonelli, C. Bloise
KLOE Memo 128−98

vi **GEANT − Detector Description And Simulation Tool,**
Application Software Group, Computers and Network Division,
CERN Program Library Long Writeup W5013

vii **HepDB, Database Management Package, Reference Manual,**
Application Software Group, Computers and Network Division,
CERN Program Library Long Writeup Q180

viii **IBM Data Management: DB2 Product Family,**
IBM Corporation,
http://www.software.ibm.com/data/db2/

ix **IBM ADSTAR Distributed Storage Manager (ADSM) Home Page,**
IBM Corporation,
http://www.storage.ibm.com/software/adsm/

x **Uniform Resource Locators (URI): Generic Syntax,**
T. Berners−Lee, R. Fielding, L. Masinter,
W3 RFC 2396

xi **The block cipher Rijndael,**
J. Daemen, V. Rijmen,
http://www.esat.kuleuven.ac.be/~rijmen/rijndael/