Message System and Data Transmission in the KLOE Data Acquisition System

A.Doria [a], P.Branchini [b], M.L.Ferrer [c], W.Grandegger [c], A.Martini [c], E.Pasqualucci [c]

[a] *INFN Sez.di Napoli, Mostra d'Oltremare Pad.20, 80125 Napoli, Italy*

[b] *INFN Sez.di ISS, Viale Regina Elena 299, 00161 Roma, Italy*

[c] *INFN Laboratori Nazionali di Frascati, 00044 Frascati, Italy*

Since the KLOE Data Acquisition System is distributed in a multi-platform UNIX environment, event data, status information and run control commands have to be transmitted over the network. Different protocols are used for the different types of information transfers in order to achieve the best performances in speed and reliability. The communication methods used in the KLOE DAQ and the results of some performance tests will be presented.

Key words: DAQ system; Communication.

1 A short Introduction to the KLOE DAQ Architecture

The KLOE experiment [1], being built at the new Φ factory DAΦNE in Frascati, is planned to run at the beginning of 1998; it will perform CP-violation studies at sensitivity of the order of 10^{-4}. The data rate from the detector at full luminosity is estimated to 10^4 events/s, corresponding to a total throughput of 50 MBytes/s. With these requirements, fast electronics, fast data transmission and efficient on-line software had to be designed for KLOE Data Acquisition.

The KLOE DAQ system uses a FDDI GIGAswitch to connect 10 readout chains to 10 farms of processors. In the readout chains - through custom modules and buses - the data coming from the detector pass a two level concentration; a commercial VME-CPU board in each chain is in charge of collecting packets of sub-events and sending them through the switch to a given processor in a farm. The packets of sub-events related to the same trigger number have to be sent to the same farm, where the complete event is build. The correspondence between data packets and event building processors is dynamically managed by the Data Flow Control (DFC), that optimises the traffic on the switch ports and the load of the farm processors.

In the described architecture network functionalities are crucial for the system operation. While standard TCP/IP is used for data transmission, network management and "inter-node" communication (from now on we will call "node" each DAQ processor) are implemented using the Simple Network Management Protocol . The standard SNMP [2] is based on reliable UDP and is widely used to manage network devices. Information about the managed objects is represented as tree of variables in a Manager Information Base (MIB), where variables can be retrieved or set by a SNMP daemon running on the remote device. Extensions of the standard MIB allow to manage private information.

2 The Message System

The commands issued by the Run Control and the requests by monitoring and debugging tasks are distributed in the system in the form of messages, with command acknowledge mechanisms. All the programs taking part to the KLOE DAQ comply with a common skeleton, that implements the message system and the storage of relevant information in a shared memory.

Inside a single node the inter-process communication is implemented, according to the UNIDAQ [3] model, with standard UNIX facilities: each process has a message queue, where commands are queued by the sender process, and a signal is also sent to inform the receiver of command arrival. The message queue number is hold in a shared memory, where relevant parameters for each process running on the node are registered.

To carry out communication between remote nodes, a "command server" implemented as a SNMP daemon runs on each node; the standard MIB has been extended to introduce the KLOE subtree, where the command server maps all the variables held in the local shared memory. This makes information about each local process accessible from the network.

Let's have a closer look to the way of sending commands to remote processes and to get two levels of acknowledgement. The sender transmits its communication request to the command server of the node where the destination process is running, locating in this way the MIB variables corresponding to the destination process.

The message is forwarded to the local process by the command server, with the procedure previously described for communication inside the single node.

The first level of acknowledgement means that the message has been received by the process: it is obtained when the destination process itself writes the received command string in a variable stored in shared memory. In fact the sender process continues querying this value to the command server, until it corresponds to the sent message. Then the sender begins to require to the command server the second level of acknowledgement, which is got when the destination process completes the execution of the command and updates a state value in the shared memory to "success" or "error". Opportune timeouts are set to assure the communication termination in case of faults.

A command syntax has been defined that allows the Run Control to send a message to a list of destinations, in a sequential or parallel mode, depending if the execution of the command by the different processes in the list is correlated or independent. In the first case the command is sent to the next process in the list only when the previous one has acknowledged a successful execution. In the case of parallel mode, the message is dispatched to all the destinations in the list, command execution takes place in parallel and a global acknowledge is obtained when all processes have completed their executions.

Performance measurements have shown that less then 30 ms are needed to send a remote command and get all the related acknowledgements, while 1.2 ms is needed if the command is sent locally (on a HP 900/735 processor).

3 The Data Transmission

The flow of event data in our system, as described in the introduction, is shown with more details in Fig. 1. In the second level of the readout chain, the "collector" process reads from a FIFO the sub-event data coming from the electronic modules and builds packets containing a fixed number of sub-events. These streams are distributed in a multiple circular buffer, where each buffer has a defined destination assigned by the DFC [4].

The circular buffer allows to pass event data between two processes with very simple mechanisms and avoids unnecessary data copies; in particular a multiple circular buffer behaves as a parallel pipe between processes. The "sender" is the level-2 process in charge of opening the TCP/IP connections with the destination farm nodes, extracting the sub-event streams from the parallel pipe and sending the packets in parallel through the FDDI GIGAswitch.

On each farm node a "receiver" process runs, that accepts the TCP/IP connections from the senders and gets sub-event packets in parallel, filling another parallel pipe, from which the process that performs event building takes its input.

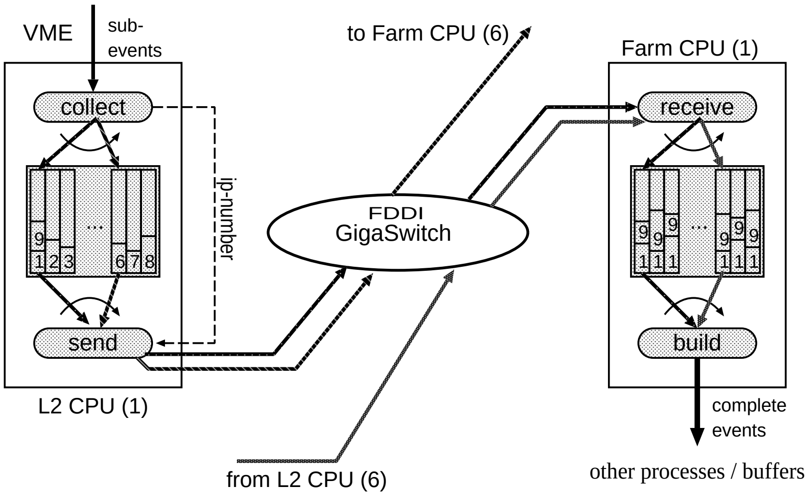

Fig. 1. Event Data Flow in the KLOE DAQ

The described data transmission model has been tested in different configurations using 10 processors connected to the FDDI GIGAswitch to verify the performances of TCP/IP event transfer, the send/receive software and the switch behaviour under load. The test environment was very heterogeneous (HP/HP-UX, Alpha/OSF1, PowerPC/LynxOS) which represent a bad case for TCP/IP communication.

Only one readout chain was available, so collectors producing random sub-events have been used for the tests with N senders and N receivers. With an estimated event size of 5 KBytes, packing 100 sub-events per stream, the average length per chain is simulated to be 50 KBytes. Using optimised TCP/IP parameters in a 4 (senders) $\times$ 4 (receivers) configuration, a throughput of $\sim$ 6.8 MBytes/s per each receiver was achieved which is compatible with our requirements. Performing the same test in 1 $\times$ 1 and 1 $\times$ 8 configurations, we verified that N $\times$ N configuration does not put additional load on the system compared to 1 $\times$ 1 and that 1 $\times$ 8 slightly increases the throughput.

References

[1] The KLOE collaboration, *KLOE, a general purpose detector for DAΦNE*, **LNF-92/019** (1992);

[2] See Documentation list on *The Simple Times*, **Vol.4, N.1**, January 1996;

[3] R.Ball (ed.), *UNIDAQ documentation Set V2.2*, **SDC-93-573**;

[4] E.Pasqualucci et al. *Using SNMP implementing Data Flow and Run Controls in the KLOE experiment*, **DAQ96 Workshop**, Osaka November 1996.