Proceedings of ICRC 2001: 2896 © Copernicus Gesellschaft 2001

ICRC 2001

The central data acquisition system for the H.E.S.S. telescope system

C. Borgmeier, K. Mauritz, and C. Stegmann, for the H.E.S.S. Collaboration

Humboldt University Berlin, Invalidenstr. 110, D-10115 Berlin, Germany

Abstract. The paper gives an overview of the central DAQ system of the H.E.S.S. telescope system. Special emphasis is given to the software design.

1 Introduction

The High Energy Stereoscopic System, H.E.S.S. is a next generation imaging air Cherenkov telescope designed to provide a comprehensive study of non-thermal phenomena in the universe. The experiment will consist of a four telescope array in the first phase which is currently under construction in the Khomas Highland of Namibia. A detailed status report of the H.E.S.S. experiment is given elsewhere (Hofmann, 2001).

Each telescope in the array is a heterogeneous system with several subsystems, including a camera with 960 individual photo-multiplier tubes (pixels), light pulser systems for pixel calibration and flat-fielding the camera, a tracking system for telescope movement, and two CCDs for guide star observation and mirror alignment. Other subsystems in the array include the PC farm, IR radiometers, cloud scanners, and optical telescopes for controlling the atmospheric properties.

The Data Acquisition System (DAQ) is a crucial part of the experiment, which provides the communication between all of these systems, as well as the run control, slow control, data collection, error handling, and monitoring to all subsystems.

2 Requirements

The main data flow is the event information from each camera to the farm. The signals of each pixel of a camera are read out via a high gain and a low gain path with 10 bits dynamic range each. The event size from a camera is 4 kB, which with data reduction reduces to approximately 1.5 kB. Hence, the expected trigger rate of the four telescope system of 1 kHz yields a data rate of 6 MB/s. A typical observation night will result in approximately 100 GB of event data to be handled by the DAQ system.

Additionally, the subsystems produce data of different sizes and at different rates. Data rates vary from a few bytes every fraction of a second from the tracking control system up to several kbytes on a minute time scale for image readout from the CCDs. The DAQ system provides tools to collect, store, display, and link monitor data to the event data.

The H.E.S.S. experiment site in the Khomas Highland in Namibia has only a small bandwidth telecommunication connection to the participating institutes, so only minor intervention from the outside is possible. Thus, the DAQ system must be designed in a robust fashion and easily operated. It is also expected that the H.E.S.S. telescope system will expand beyond Phase I, which demands that the DAQ system be flexible to incorporate new components easily.

3 The Central DAQ Hardware

The central DAQ consists of a Linux PC farm connected by a 1 GBit ethernet swcitch, the layout of which is shown in Figure 1. The PCs contain dual Pentium III processors with a 800 MHz clock and 256 MB RAM. Two servers, similar to the farm PCs but with expanded networking capabilities, control a 640 GB Raid array and two DLT tape-drives. The front-end computers on the telescopes are connected to the farm via a 100 Mbit/s fiber optic network. The whole system is synchronized by a GPS time server to better than the required accuracy of 1 ms.

The construction of the farm from discrete commercial components allows maintainability and upgrades of the central DAQ. Figure 2 shows a photograph of the central DAQ during tests in Berlin. The left and right racks contain the 16 farm PCs and the middle rack contains the other components.

During data-taking, each camera CPU sends its event data for a defined period to a single farm processor. While the data on this PC are processed and stored locally in files (see 4.2), data are sent to the other PCs in turn. The monitor data

Correspondence to: C. Stegmann (stegmann@ifh.de)

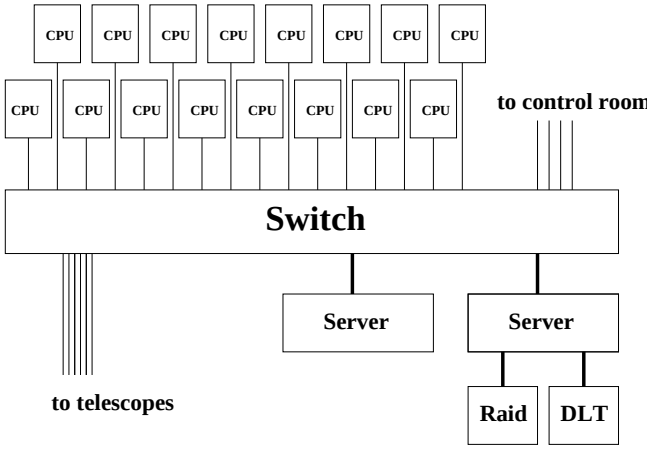

Fig. 1. The layout of the central DAQ system.

Fig. 2. The central DAQ system during test in Berlin.

are collected centrally and also stored, and those data needed for event processing are distributed concurrently to the farm CPUs. In the morning, after the data have been collected, they are read back from the farm PCs, assembled in a single chronological event stream, and written to tape.

4 DAQ Software

The software for the central DAQ follows an object-oriented design approach to provide flexibility and maintainability. The central DAQ software package, DASH (Data Acquisition Software for H.E.S.S.), provides basic tools to build various on-line processes. DASH makes use of another software package, SASH (Storage and Analysis Software at H.E.S.S.), to provide classes for data storage and analysis.

4.1 DASH

The DAQ system needs to interface to the various subsystems for control, configuration, error-handling, monitoring, and read-out. A limited set of states has been defined to control the system. DASH needs to communicate with these diverse systems and ensure state-consistency. The DASH library provides simple, effective tools for their control, configuration, and data transfer.

The fundamental DASH design to provide flexibility and consistency throughout all subsystems is a building block architecture. This permits systematic construction of various subsystem controllers, like the tracking controller, CCD controller, run controller, monitor servers, farm manager, etc.. These subsystem controllers interact to manage the H.E.S.S. array.

The building blocks are C++ classes which the actual controllers build upon from by *inheritance* and *aggregation*. Each building block provides a communication interface and functionality of state control and data flow. The Inter Process Communication and the major building blocks are described below.

4.1.1 Inter Process Communication

The DASH Inter Process Communication (IPC) uses *omniORB*, an open source communication package developed by AT&T according to CORBA specifications (OmniORB, 1998). CORBA is a distributed object-oriented client-server platform, which includes a Remote Procedure Call (RPC) mechanism, object services, such as Naming Service, and language mappings for different programming languages. CORBA allows remote objects to be transparently accessed as if they were local.

In addition, DASH uses the *omniORB* Name Server to register all running DASH objects. The Name Server permits objects to be referenced by name without explicit knowledge of the host computer and its port number. Object references can be structured in the Name Server similar to a Unix directory structure.

4.1.2 Server and State Controller

Each subsystem controller is a remotely accessible server with well-defined state transistions. DASH provides the classes, `Dash::Server` and `Dash::StateController`, as building blocks to construct these. The `Dash::Server` class calls transparently all necessary methods to register the object at the name service and readies the IPC. It provides common methods for all DASH server applications so that classes deriving from it are only required to implement their specific methods. The `Dash::StateController` class, which maintains the current state of an object and manages its state transitions. It provides the common interface of all actual servers participating in the system control.

4.1.3 Configuration

The configuration of each subsystem is organized by a central *MySQL* database. This allows the retrieval of parameter change histories. A special H.E.S.S. configuration table format, including a C++ interface, has been developed to permit

database access without explicit SQL knowledge.

4.1.4 Data Transfer

The transfer of any data inside the DAQ system follows a push-architecture. This means that the data source decides when to send data while the destination is always prepared to accept them. Each source knows the target where the data are to be sent. DASH includes the building block class, `Dash::BlockAcceptor`, to provide the interface of each link in the data chain. Its main method, the `TakeBlock` function, accepts a data block. The block acceptor can maintain a pointer to a target to which the block is then forwarded.

Among the DASH-provided specializations of the block acceptor are the `Dash::Sender` and `Dash::Buffer`. The former connects to a remote destination and synchronously forwards all accepted data to it. The latter buffers the accepted data blocks and sends them asynchronously.

The combination of these building blocks allow different synchronous and asynchronous data transport networks. A data source connected to a sender object uses a synchronous blocking data transport, i.e. the control returns as soon as the data has been sent. An intermediate buffer object decouples the source from the network transport and allows an asynchronous data transport with a minimal latency for a control process.

The actual work on the data block is left to specializations by derived classes. DASH provides further block acceptor specializations which are able to understand and process different types of data blocks.

4.1.5 Monitoring Data

Monitoring data transfer is a specialization of the data transfer and can be handled like any of the above mentioned data blocks when transported between objects. The SASH library (see 4.2) provides a hierarchy of monitor classes whose common feature is a time stamp. This allows the subsequent recombination of data between loosely connected systems.

SASH has defined classes for the basic monitor information for each subsystem. Additionally, SASH provides a dynamic monitor class, which contains a map of numbers or object pointers, each one indexed by a qualifying name. All monitor events are stored in ROOT (ROOT, 1996) files and are accessible for later analysis. Additionally, the monitor information can be displayed on-line using ROOT graphics.

4.2 SASH

SASH is based upon ROOT and is responsible for data storage and both on-line and off-line analysis. The requirements for such a package include the flexiblity to adapt to changes in which telescopes are used for different runs during a night, changes (i.e. perhaps even different cameras) during the lifetime of the experiment, and the speed to be able to function during the on-line data-taking. The flexibility is supported by the object-oriented ROOT framework. ROOT also provides a C++ interpreter for interactive work, the capability of dynamic library linkage, and object persistency. That is, any object derived from a ROOT base class can be saved in a ROOT file. The analysis tools in ROOT support physics analysis with diagram, histogram, and fitting features.

SASH is a collection of C++ classes of distinct types: data "containers" and data "manipulators". The former stores the data into various objects that form a representation of the actual system. For example, a telescope object maintains the telescope's configuration, monitoring information, and event data. It also contains all of its pixel objects, each of which has access to all data and information about itself. The "manipulator" type of class is based on the *ROOT Maker* concept and is capable of manipulating the data and writing it into the objects. It also stores its own analysis parameters into the data file, which is especially useful for reproducing the original results.

The data are converted on-line into the SASH format using the ROOT tree concept for all types of event storage. ROOT trees store data in a tabular format that is then mapped onto objects. The raw data, each type of monitoring data, and many possible instances of the processed data are each stored in their own tree. In this way, the "DST" can be user-specific, depending on what trees and which parts of the trees the user chooses to keep. Storing the reconstructed parameters during an analysis in these trees also grants SASH extra flexibility for comparison of various cuts and algorithms.

5 Performance

The maximum reachable bandwidth of the central DAQ system has been tested by sending simulated telescope events simultaneously from four farm PCs, acting as senders, to a single farm PC, acting as receiver. The sender and receiver were based on the software described above, using a CORBA protocol for data transfer. The event size was varied to analyze the behavior of the system from 100 bytes to 4 kB and the achieved data rate was measured. The bandwidth was found to be equally shared between all senders. The maximum input data rate into the receiver as a function of the event size is shown in figure 3. The data rate exceeds the required bandwidth of 6 MB/s for event sizes larger than 300 Bytes. For the expected event size of a single camera event of approximatly 1.5 kByte the bandwidth is 11 MB/s, yielding a safety margin of nearly a factor two.

Monte Carlo events have been used to test the performance of writing events into the SASH tree structure at the farm. The raw event size was found to be within expectations and the rate for reading and writing the events ranged from between 100 Hz to 300 Hz per farm node, depending on the choice of on-line data compression. With the expected trigger rate of 1 kHz, writing events will take only a fraction of the available computing capacity.

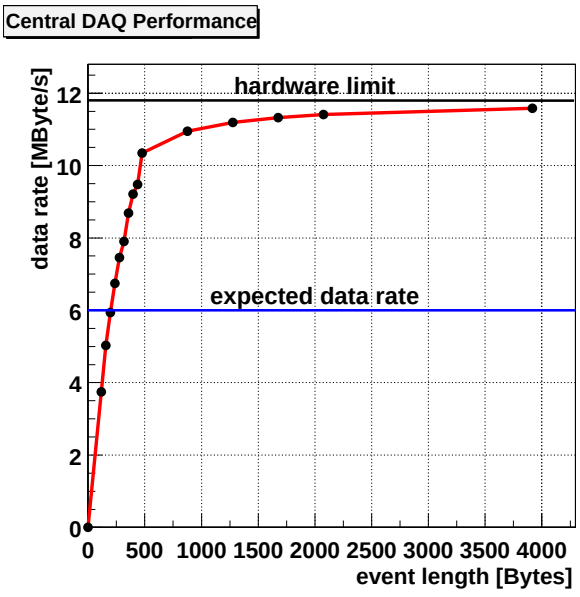

Fig. 3. Performance of the central DAQ system.

6 Summary

The central DAQ system of the H.E.S.S. telescope array consists of a modular high performance PC farm. Special emphasis was given to the development of a flexible and robust software system based on C++, CORBA, and ROOT. By the time of the conference, the DAQ system will be installed and in operation at the site in Namibia.

Acknowledgements. This work was supported by the Bundesministerium für Forschung und Technologie under the contract number 05 AS9KHA 6.

References

W. Hofmann, 2001, Proc. of the 27th ICRC, Hamburg, 2001, to be published

S. L. Lo and S. Pope, The Implementation of a High Performance ORB over Multiple Network Transports, Distributed Systems Engineering Journal, 1998. See also `omniORB: http://www.uk.research.att.com/omniORB-/omniORB.html`

R. Brun and F. Rademakers, ROOT – An Object Oriented Data Analysis Framework, Proceedings AIHENP'96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. & Meth. in Phys. Res. A 389 (1997) 81–86. See also `http://root.cern.ch/`.