DEVELOPING A NEW DAQ SOFTWARE FOR THE COMPASS EXPERIMENT

Vladimír Jarý[1], Tomáš Liška, Miroslav Virius

Czech Technical University in Prague, Faculty of Nuclear Sciences and Physical Engineering
Vladimir.Jary@cern.ch, Tomas.Liska@cern.ch, Miroslav.Virius@cern.ch

ABSTRACT:

COMPASS is a particle physics experiment situated on the SPS accelerator in the CERN laboratory. The existing data acquisition system of the experiment uses standard servers and the DATE software. With the increasing data rate and the age of the hardware, the failure rate and the dead time of the system is also considerably increasing, thus a new DAQ system based on a custom hardware is being developed.

At first, the current DAQ system is discussed, and then the overview of the software for the new system is analyzed. Finally, the DIM library that is used for communication between nodes is presented and benchmarked.

KEYWORDS:

COMPASS, data acquisition, DATE, networking

INTRODUCTION

Modern particle physics experiment produces enormous quantities of data. It is not possible to analyze data online, therefore data acquisition systems are used to select, readout, digitize, filter, and store physically interesting events. This paper focuses on the data acquisition system of the COMPASS experiment.

At first, the physical program of the experiment is briefly introduced. COMPASS uses the ALICE DATE package for the data acquisition (DAQ), thus this package is described in more details. DATE software runs on the standard x86-compatible hardware. The dead time of the system increases with increasing trigger rate, thus a development of the custom FPGA-based hardware for DAQ has started. It has been decided to replace the DATE package with brand new software.

COMPASS EXPERIMENT

The COMPASS, which stands for the Common muon and proton apparatus for structure and spectroscopy, is a fixed target experiment running at the Super proton synchrotron (SPS) particle accelerator at the CERN laboratory in Geneva, Switzerland [1]. The scientific program, which includes experiments with the hadron and the muon beam, has been approved by the CERN scientific council in 1997. After several years of preparations, construction, and commissioning, the data taking started in 2002. Currently, a proposal for the second phase of the experiment (see COMPASS-II [2]) has been submitted to the scientific council for approval. The second phase would continue at least until 2015, if approved by the council.

[1] Corresponding author

When the beam particles interact with the polarized target, secondary particles are produced. These secondary particles are detected in a system of detectors that form the COMPASS spectrometer (see Figure 1). Detectors are used to track particles, to identify particles, and to measure energy of particles. Particle tracking is performed mainly by various wire chambers. Particle identification is implemented by the muon filters and the Cherenkov detectors. Finally, the deposited energy is measured by the electromagnetic (for electrons and photons) and the hadronic (for hadrons) calorimeters. Set of data describing the flight of particles through the spectrometer is known as an event. Size of the typical event amounts to 35 kB.

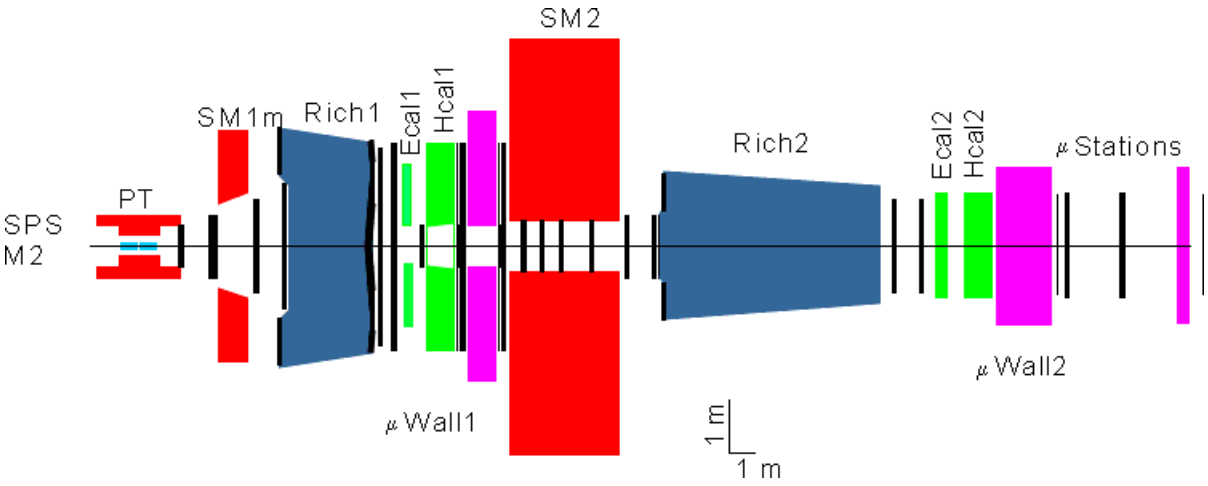

Figure 1: The layout of the COMPASS spectrometer, image taken from [2]

The SPS accelerator works in 16.8 s long cycles that consists of 12 s of acceleration and 4.8 s of particle extraction. The extraction period is also known as the spill (burst). The cycle of the accelerator has strong impact on the design of the DAQ system.

CURRENT DATA ACQUISITION SYSTEM

The DAQ system used by the COMPASS experiment consists of several layers [9][12]. On the lowest level, the detector frontend (primary) electronics lie. The primary electronics continuously preamplify, discriminate, and digitize data from detectors. There are approximately 250000 data channels; data streams from multiple channels are concentrated into the concentrator modules called GeSiCA and CATCH. The GeSiCA (GEM and Silicon Control and Acquisition) modules serve the GEM and silicon detectors that are characterized by a high data rate. The CATCH (COMPASS Accumulate, Transfer and Control Hardware) modules perform the readout of the other detectors. Although the frontend electronics constantly produce data, the readout by the concentrator modules is performed only when the trigger control system TCS selects physically interesting event (the decision is based on the amount of deposited energy in calorimeters and on the signals from hodoscopes). Additionally, the TCS distributes the timestamp and the event identification. Concentrator modules create data blocks called subevents by assembling data from multiple detector channels corresponding to the same trigger and appending the subevent header with metainformation provided by the TCS.

The subevents are transferred into the next processing stage, the readout buffers (ROB), using the optical link S-Link. ROBs are standard servers equipped with 4 spillbuffer PCI cards. Each spillbuffer has 512 MB of onboard memory which serves as a buffer for incoming data: it is being filled during bursts and it is being continuously unloaded into the main memory (RAM) of the ROB. The ROB layer uses the cycle of the SPS accelerator to reduce the data rate to the higher processing stages to one third of the onspill rate. From the ROB layer, the subevents are pushed into the next stage which is composed of computers called event

builders (EVB). ROBs and EVBs are connected into the Gigabit Ethernet network. Event builders use the information from the subevent headers to create full events and catalog files with metainformation. The catalog files are stored into the ORACLE database, the events are sent to the CERN tape permanent storage CASTOR after some delay. Remaining CPU power of the event builders is dedicated to the event sampling and event filtering.

The DAQ system consists of the custom hardware and the industry standard components. For example, the spillbuffer PCI cards have been developed at the Munich Technical University for the COMPASS experiment; the S-Link technology has been developed for the ATLAS experiment. On the other hand, ROBs and EVBs are standard, x86 compatible servers.

DATA ACQUISITION SOFTWARE

The COMPASS DAQ software is based on the modified DATE (Data Acquisition and Test Environment) package [3] maintained by the collaboration of the ALICE experiment. The DATE functionality includes data flow control, run control, event sampling, interactive configuration, or information reporting. The system configuration and software logs are handled by the MySQL database [5][7]. Additionally, the DATE also defines the data format that represents events and subevents. The DATE is very scalable system; it can perform data acquisition on the small laboratory systems with only one computer, on the other hand it runs also at the ALICE experiment with hundreds of computers.

The DATE is written (mainly) in the C language; the requirements on the DAQ hardware are following: each processor involved in the system must be x86–compatible, each processor must be running (32-bit) GNU/Linux operating system, and all processors must be connected to the same network and must support the TCP/IP stack.

The DATE divides processors involved in the DAQ into two main categories: the LDCs and the GDCs. The LDC (Local Data Concentrator) performs the readout of event fragments. Depending on the configuration, the subevents are either saved directly on the LDC, or send over the network to some GDC. The GDC (Global Data Collector) performs the event building. The DATE supports load balancing of GDCs through the Event Distribution Manager (EDM) agent. If the EDM is not used, event fragments are distributed to GDCs using a round robin algorithm. Clearly, the LDCs correspond to the readout buffers and the GDCs to the event builders in the COMPASS terminology.

Several additions to the DATE package have been implemented for the needs of the COMPASS experiment. E.g., the program COOOL (COMPASS Object Oriented Online) is analyzing part of the data online on one event builder. The results presented in a form of the ROOT histograms are used by the shift crew to monitor the detector performance. To reduce the amount of events stored to the tapes, the online filter program, called Cinderella [11], has been developed. Online filter can be regarded as a high level trigger, it rejects physically uninteresting events. Electronic logbook is a tool used to store metainformation about runs and spills. Most of the information (e.g. the run list) is added automatically by the DATE system but a shift member can also add comments manually. The logbook is stored in the MySQL database; users can browse it using the web interface based on the PHP language.

NEW DATA ACQUISITION ARCHITECTURE

During the first year of the data-taking (2002), 260 TB of data have been recorded. During the 2004 Run, the DAQ system recorded 508 TB of data [1][9][11][12]. The increase is caused by increased number of detector channels and increased trigger rate. However, increasing the trigger rate also increases the DAQ dead time. The dead time is defined as a ratio between the time when the system is busy (i.e. it cannot process any new events) and the total time. The dead time increased to 20% in the 2010 Run, this means that one fifth of the beam time was

wasted. Additionally, as the hardware gets older, the failure rate also rises. A research and development of the new DAQ architecture that would overcome the aforementioned problem has started.

The new DAQ system would perform the readout and the event building by a dedicated custom hardware based on the Field Programmable Gate Array (FPGA) integrated circuits [10]. This would greatly reduce the number of components involved in the data acquisition and consequently, the reliability would be increased. Additionally, the existing readout buffers and event builders could be used for another tasks, e.g. for the filtering of events. Since the event building would be done by the hardware, the run control and the monitoring would be the main tasks of the DAQ software.

We have evaluated the possibility of using the DATE package for the new DAQ system [8]. The DATE is too complex software for the needs of the proposed architecture. Additionally, DATE requires x86 compatible hardware. Thus, it has been decided to develop custom control and monitoring software. On the other hand, some components (e.g. a logbook) of the DATE package could be reused after small modifications. Furthermore, the data structures used for the events must remain unchanged because of the compatibility with programs for the offline analysis. Also the compatibility with the detector control system DCS must be retained. The DATE represents the entities involved in the DAQ (ROBs, EVBs, EDM, …) by the finite state machines implemented in the State Management Interface (SMI++) framework [6]. The communication between the state machines is based on the Distributed Information Management (DIM) library [4] which is included in the SMI++ framework. The DIM also intermediates the communication with the DCS system, therefore the DIM library should be used in the new DAQ software.

EVALUATION OF THE DIM LIBRARY

Originally, the DIM has been developed for the needs of the DELPHI experiment at CERN, today it is used at LHC experiments. The DIM library provides functionality for the asynchronous, one to many communication in the heterogeneous network environment. Based on the TCP/IP, it is running under GNU/Linux, Windows, Solaris, Darwin, VMS, VxWorks, and other operating systems. Interfaces for C, C++, Java (through Java Native Interface), FORTRAN, and Python languages are available. The communication system consists of the DIM name server (DNS), the publishers (servers), and the subscribers (clients).

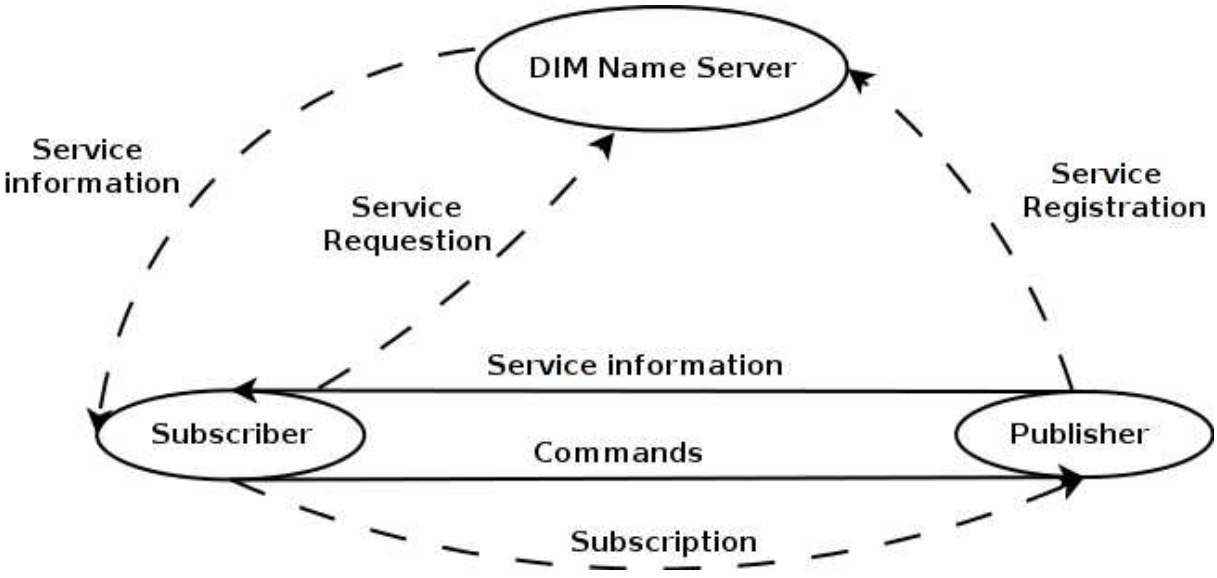

Figure 2: The functionality of the DIM-based communication according to [4]

Information services are characterized by a unique name. When a publisher wishes to publish a new service, it contacts the DIM name server (see Figure 2). The DNS registers the service name, the service format, and the address of the publisher. When a subscriber wishes to subscribe to some service, it sends request with the service name to the DNS which returns the address of the corresponding publisher. In the application code, it is not necessary to explicitly communicate with the DNS; everything is done transparently provided that the environment variable DIM_DNS_NODE that contains the address (either IP or hostname) of the node with the DNS is properly set. Additionally, the DIM handles conversion of data from the host encoding to the network encoding. Thus, using the DIM to implement the communication is extremely simple. The DIM services can be divided into 3 categories:

1. services that are requested by a subscriber only once
2. services that are updated regularly at given interval
3. services that are updated when a monitored quantity changes

Additionally, the clients can also send DIM commands to the servers. The following C++ code implements a sample command service:

```
class TestCommand: public DimCommand{
  void commandHandler(){
    cout << "Received command: " << getInt() << endl;
  }
public:
  TestCommand(): DimCommand("TEST_COMMAND", "I");
};
```

To create a command service, one needs to subclass the `DimCommand` class and overload the `commandHandler` method. This method is called whenever a subscriber sends a command. In the constructor, a super class is created: the first parameter represents the unique service name that will be used to identify the command service by the DNS and by subscribers. Second parameter specifies the message format: the letter "I" stands for integer, the letter "F" for float, and the letter "C" for character. It is possible to combine the letters to define structured messages. Typically, in the `commandHandler` method, the sent command data is received and processed. To receive an integer message, one can use the `getInt` method. There are similar methods that receive floats and characters. A structured message can be obtained by the `getData` method that returns a void pointer. To actually use the command service, an instance of the `TestCommand` class must be created and the DIM server must be started:

```
int main(int argc, char **args){
  TestCommand command;
  DimServer::start("TEST_SERVER");
  while(true){ pause(); }
}
```

The command service is running in the separate thread, therefore it is possible to pause the execution of the main thread. The code that implements a subscriber of this command service is even simpler:

```
int main(int argc, char **args){
  const int START_RUN = 1;
  DimClient::sendCommand("TEST_COMMAND", START_RUN);
  return 0;
```

```
}
```

The command is sent by calling the static method `sendCommand` of the `DimClient` class. First parameter identifies the command service; the other is the value that should be sent to the service.

We have measured the performance of the DIM library in order to discover the optimal message size. The test case consists of a publisher that publishes a command service and one monitored service. A subscriber sends a command. When a publisher receives the command, it updates the monitored service which triggers a subscriber to fetch the updated value. When a subscriber receives the update message, it sends another command to a publisher. When this cycle is repeated million times, the average data flow and number of exchanged messages per seconds are calculated.

Size of the message	Data flow [kB/s]	Received messages [s^{-1}]	Size of the message	Data flow [kB/s]	Received messages [s^{-1}]	Size of the message	Data flow [kB/s]	Received messages [s^{-1}]
4 B	14	3700	256 B	923	3700	16 kB	8840	600
8 B	29	3700	512 B	1582	3200	32 kB	10390	300
16 B	58	3700	1 kB	3690	3700	64 kB	10667	170
32 B	116	3700	2 kB	3899	1900	128 kB	11035	90
64 B	233	3700	4 kB	7246	1800	255 kB	11179	40
128 B	456	3700	8 kB	7407	900			

Table 1: Results of the DIM performance

The results for different sizes of messages are summarized in the Table 1. The measurements have been performed on the local 100Mbit network, thus the maximum theoretical throughput is 12800 kB/s. As the message size increases, the achieved data rate approaches the maximum value. Some bandwidth is consumed by the overhead of the TCP/IP, some by the communication with the DNS. The relation between data flow and the message size is linear for the smaller messages (roughly until 4 kB), for the larger messages, the increase in the data flow is slower as the flow converges to maximum theoretical value. Number of exchanged messages remains constant for messages smaller than 1 kB. The same tests have been measured for the communication through the loopback device. In this case, the system exchanged 23000 messages per seconds if the message size was smaller than 8 kB, the maximum data rate of 260 MB/s was reached for 16 kB messages. Our results seem to be in a good accordance with a similar measurement published by C. Gaspar[2].

OUTLOOK

We have defined a test case that simulates a simple DAQ system. The system consists of one master and several slave processes that are running on the distributed machines. The information about slave processes involved in the system should be stored in the database or in the XML file. The master process fetches this information and initializes all slave processes. The master regularly broadcasts messages to all slaves; the slaves return the confirmation message. The latency should be measured for different sizes of messages, for different number of slaves, for different message frequencies. The test case should be used to compare suitability of the C++, Java, and Python languages for developing the new DAQ software. The test case should be evaluated during May 2011.

[2] See http://dim.web.cern.ch/dim/DIM_Performance.pdf

Later in this year, the first test runs with the new hardware will be carried out, thus at least minimal run control software will be required.

CONCLUSSION

The existing data acquisition system of the COMPASS experiment has been described. The system suffers from a high dead time caused by the recent increases in the trigger rate. Development of the new DAQ system based on a custom hardware has started. The new system will require a new run control and monitoring software. The system will use the DIM library to ensure the compatibility with other software systems of the experiment. The DIM library has been tested; the message size should not excess 1 kB if the highest frequency of exchanged messages is required. In the near future, the suitable programming language will be chosen and the development of the run control application will begin.

ACKNOWLEDGEMENT

This work has been supported by the MŠMT grants LA08015 and SGS 11/167.

BIBLIOGRAPHY

[1] P. Abbon et al. (the COMPASS collaboration): *The COMPASS experiment at CERN*, In: Nucl. Instrum. Methods Phys. Res., A 577, 3 (2007) pp. 455–518. See also the COMPASS homepage at http://wwwcompass.cern.ch

[2] Ch. Adolph et al. (the COMPASS collaboration): *COMPASS-II proposal*, CERN-SPSC-2010-014; SPSC-P-340 (May 2010)

[3] T. Anticic et al. (ALICE DAQ Project): *ALICE DAQ and ECS User's Guide*, CERN EDMS 616039, January 2006

[4] P. Charpentier, M. Dönszelmann, C. Gaspar: *DIM, a Portable, Light Weight Package for Information Publishing, Data Transfer and Inter-process Communication*, Available at: http://dim.web.cern.ch

[5] L. Fleková, V. Jarý, T. Liška, M. Virius: *Využití databází v rámci fyzikálního experimentu COMPASS*, In: Konference Tvorba softwaru 2010, Ostrava: VŠB - Technická univerzita Ostrava, 2010, ISBN 978-80-248-2225-9 pp. 68–75.

[6] B. Franek, C. Gaspar: SMI++ State Management Interface [online]. 2011. Available at: http://smi.web.cern.ch

[7] V. Jarý: *COMPASS Database Upgrade*, In: Workshop Doktorandské dny 2010, Prague: Czech Technical University in Prague, Czech Republic, November 2010, ISBN 978-80-01-04644-9, pp. 95–104

[8] V. Jarý: *DATE evaluation*, In: COMPASS DAQ meeting, Geneva, Switzerland, 29 March 2011

[9] A. Král, T. Liška, M. Virius: *Experiment COMPASS a počítače*, In: Československý časopis pro fyziku 2005, č. 5, str. 472.

[10] A. Mann, F. Goslich, I. Konorov, S. Paul: *An AdvancedTCA Based Data Concentrator and Event Building Architecture*, In 17th IEEE-NPSS Real-Time Conference 2010, Lisboa, Portugal, 24–28 May 2010

[11] T. Nagel: *Cinderella: an Online Filter for the COMPASS Experiment*. München: Technische universität München, January 2009.

[12] L. Schmitt et al.: *The DAQ of the COMPASS experiment*, In: 13th IEEE-NPSS Real Time Conference 2003, Montreal, Canada, 18–23 May 2003, pp. 439–444