Developments and applications of DAQ framework DABC v2

J Adamczewski-Musch[1], N Kurz[1] and S Linev[1,2]

[1]Data processing group, Electronics department (RBEE), GSI Helmholtz-Zentrum für Schwerionenforschung GmbH, Planckstr. 1, 64291 Darmstadt, Germany

E-mail: j.adamczewski@gsi.de, n.kurz@gsi.de, s.linev@gsi.de

Abstract. The Data Acquisition Backbone Core (DABC) is a software framework for distributed data acquisition. In 2013 Version 2 of DABC has been released with several improvements. For monitoring and control, an HTTP web server and a proprietary command channel socket have been provided. Web browser GUIs have been implemented for configuration and control of DABC and MBS DAQ nodes via such HTTP server. Several specific plug-ins, for example interfacing PEXOR/KINPEX optical readout PCIe boards, or HADES trbnet input and hld file output, have been further developed. In 2014, DABC v2 was applied for production data taking of the HADES collaboration's pion beam time at GSI. It fully replaced the functionality of the previous event builder software and added new features concerning online monitoring.

1. Introduction

The Data Acquisition Backbone Core (DABC) is a C++ software framework that can implement and run various data acquisition solutions on Linux platforms [1]. In 2013 Version 2 of DABC has been released with several improvements. These developments have taken into account lots of practical experiences of DABC v1 with detector test beams and laboratory set-ups since first release in 2009.

The plug-in interfaces for user code and configuration procedures have been simplified. Internally the framework has been enhanced by means of smart references and object cleanup mechanisms. As a new feature, an HTTP web server component has been implemented that allows monitoring process variables, and execute commands in the DABC runtime environment.

The existing DABC data input from standard GSI PEXOR/KINPEX PCIe readout hardware has also been refactored for such framework changes. Additionally, it has been extended by a triggered data acquisition mode, emulating the behavior of the previously applied DAQ system MBS. A special web browser GUI to control this DAQ set-up has been implemented in JavaScript, using the new DABC web server access. By means of an intermediate DABC application, it is even possible to control a native MBS node from a web browser with a similar GUI.

From May to September 2014, DABC v2 was applied for production data taking of the HADES collaboration's pion beam times at GSI. It step by step replaced the functionality of the previous HADES event builder software *hadaq*. Moreover, DABC offers advanced possibilities here for online quality analysis of data samples via TCP/IP sockets, and monitoring of run variables via HTTP servers.

[2] To whom any correspondence should be addressed.

2. PEXOR/KINPEX readout

GSI standard DAQ systems often use PCI express boards PEXOR/KINPEX that receive data via optical fiber GOSIP protocol [2] from various types of front-end hardware. Corresponding drivers and DABC plug-ins had been developed [3] for control and data acquisition of such systems on Linux PCs. For DABC v2 this software has been upgraded, taking into account simplifications of the DABC framework application programmers interface (API), and implementing additional functionalities for triggered read-out and advanced set-up.

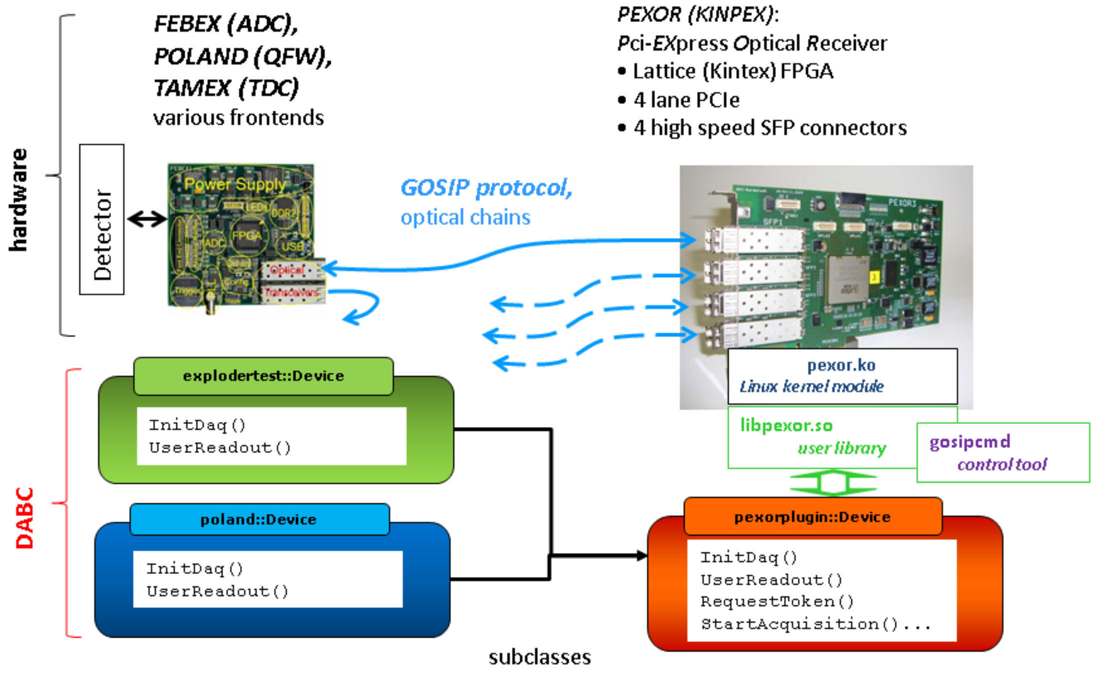


Figure 1. DABC PEXOR/KINPEX and frontend readout hardware and corresponding software components: linux kernel driver, library and command line tool provide access to PEXOR functionality. The DABC plug-ins are structured in a way that generic actions are delivered by Device base class, frontend specific actions to be handled by additional sub-class plug-ins .

Figure 1 shows a sketch of the readout hardware and the software entities. The PEXOR/KINPEX board is plugged to the PCIe slot of the readout computer with Linux operating system. Various digitizing front-end boards, e.g. ADC, TDC, or QFW, are linked to PEXOR with up to 4 chains of optical fibers. Access to the hardware from the operating system is provided by kernel module device driver *pexor.ko* with corresponding C++ library *libpexor.so.* A new command line tool *gosipcmd* allows low level configuration and register I/O concurrent to the data acquisition process. The DABC plug-in is also based on the *libpexor.so* driver library, as described in [3].

The originally monolithic DABC v.1 plug-in has now been split into a hierarchical structure with base class and well defined user subclass API. From several DAQ use-cases it turned out that any specific code would need direct access to front-end registers, because otherwise initialization at startup is not possible. Also during read out, a direct communication with frontend hardware is sometimes desirable, independent of the generic mechanisms with GOSIP token request DMA. So the natural DABC class for such partitioning is *Device*, since neither *ReadoutModule* nor *Input* (*Transport)* classes have access to the *Board* instance of the driver library [3].

The base class *pexorplugin::Device* (Figure 1) keeps all relevant methods to communicate with PEXOR and perform data read out. It implements the forwarded *dabc::DataInput* interfaces, such as *Read_Size()*, *Read_Start()*, and *Read_Complete()* [3]. By default, they handle triggered or non-triggered GOSIP token request readout, depending on the DABC setup parameters. The frontend subclasses of *Device* may even overwrite these methods to implement a different readout behavior. However, for most use cases it is sufficient to touch a few dedicated interface methods only to achieve all frontend- or experiment-specific parts. In Figure 1, for example, subclasses *explodertest::Device* and *poland::Device* will redefine virtual methods *InitDaq()* and *UserReadout()* to adjust specific functionality.

As a new feature the *pexor* driver for DABC now fully supports triggered data acquisition modes, using interrupts from the TRIXOR board attached to PEXOR. So it is possible to emulate with DABC the same read out as with the GSI production DAQ system MBS [4] at a single data receiving node, and compare mechanisms and performance of both systems. For this purpose additional virtual methods *StartAcquisition()* and *StopAcquisition()* have been implemented to *pexorplugin::Device.* Like the homonymous MBS commands, these functions by default raise special software trigger interrupts (id 14 and 15) which are evaluated in the driver layer to insert marker events to the data stream, and to change acquisition state. Moreover, they also can be extended or completely re-implemented in the *Device* subclass. Functions *StartAcquisition()*, *StopAcquisition()* and *InitDaq()* can be invoked interactively from the web control interface with corresponding DABC commands defined in *pexorplugin::Device.* Because of this any specialized frontend behavior can be controlled with generic commands.

From first tests of triggered read-out it turned out that DABC with PEXOR/KINPEX plug-in can achieve similar performance as an MBS system with identical hardware. Moreover, DABC provides another "automatic" trigger readout mode where data is acquired from the frontends already in the trigger interrupt handler and buffered in a kernel module queue before passed to the application. This mode promises even better throughput with lower latency per event, since trigger acknowledgement and subsequent data readout request does not require to be handled intermediately by the userspace application anymore. Further systematic performance tests of such readout modes in comparison with MBS are planned for this year.

3. Web server controls

Any data acquisition system requires some kind of control system that allows at least inspecting the run state of the acquisition and some variables of interest, like event building rates, amounts of stored data, or log messages. Moreover, it might be useful to change acquisition states, output file names, and other setup parameters interactively by means of such slow control command access. For DABC v1, previously the DIM or EPICS systems had been evaluated for this purpose. However, such a proprietary control protocol has the disadvantage that it needs special client software to be installed on all nodes that want to access and view the exported information. Because of this, for DABC v2 a more common interface has been intended, with GUI clients available on nearly any platform: an HTTP web server.

These developments were closely related to a similar approach for online monitoring of the ROOT analysis system [5]. Here any ROOT data object can be converted into JavaScript Object Notation (JSON) format and is accessible via an embedded HTTP server that provides a generic JavaScript visualization for any web browser. By default the hierarchy of exported objects and process variables is displayed in a tree view representation. Selecting a tree item in the browser will draw the selected object in a predefined way, or will show details of a variable. Moreover, using the full object path in the URL, it is possible to retrieve any exported object values directly from JavaScript code in the browser, or from HTTP command line tools in scripts.

The features of such a ROOT web server interface have been embedded into DABC and extended to export and visualize special DABC parameters like rate meters, and to invoke DABC command callbacks via HTTP requests. In addition to this, one DABC node (master) may also combine the object hierarchies of several distributed nodes (agents) and can make them accessible as a super-hierarchy at a single HTTP server [5]. These distributed hierarchies can be either linked to the master via dedicated DABC sockets, or via their own HTTP server, or by means of DABC plug-ins for external DAQ or control systems. Therefore one DABC webserver can act as gateway to integrate and access data objects and control variables from different sources, such as DABC, ROOT, Go4, MBS, EPICS, DIM, and FESA. As a consequence, the user could monitor both the data acquisition controls, and the online analysis results in a web browser at the same HTTP server.

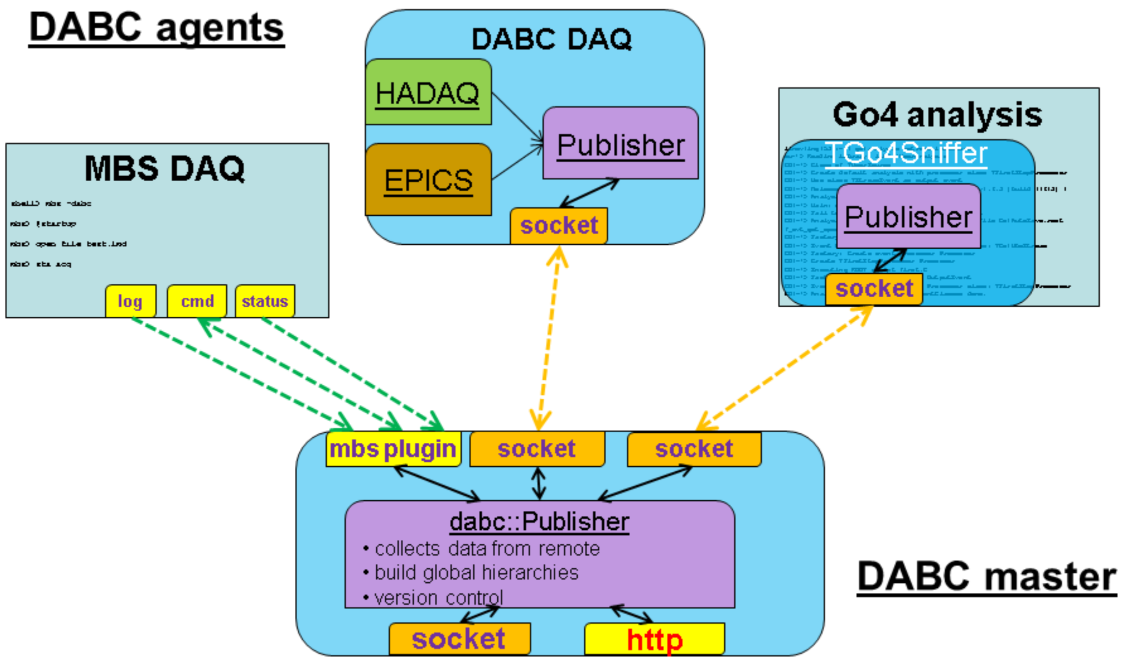
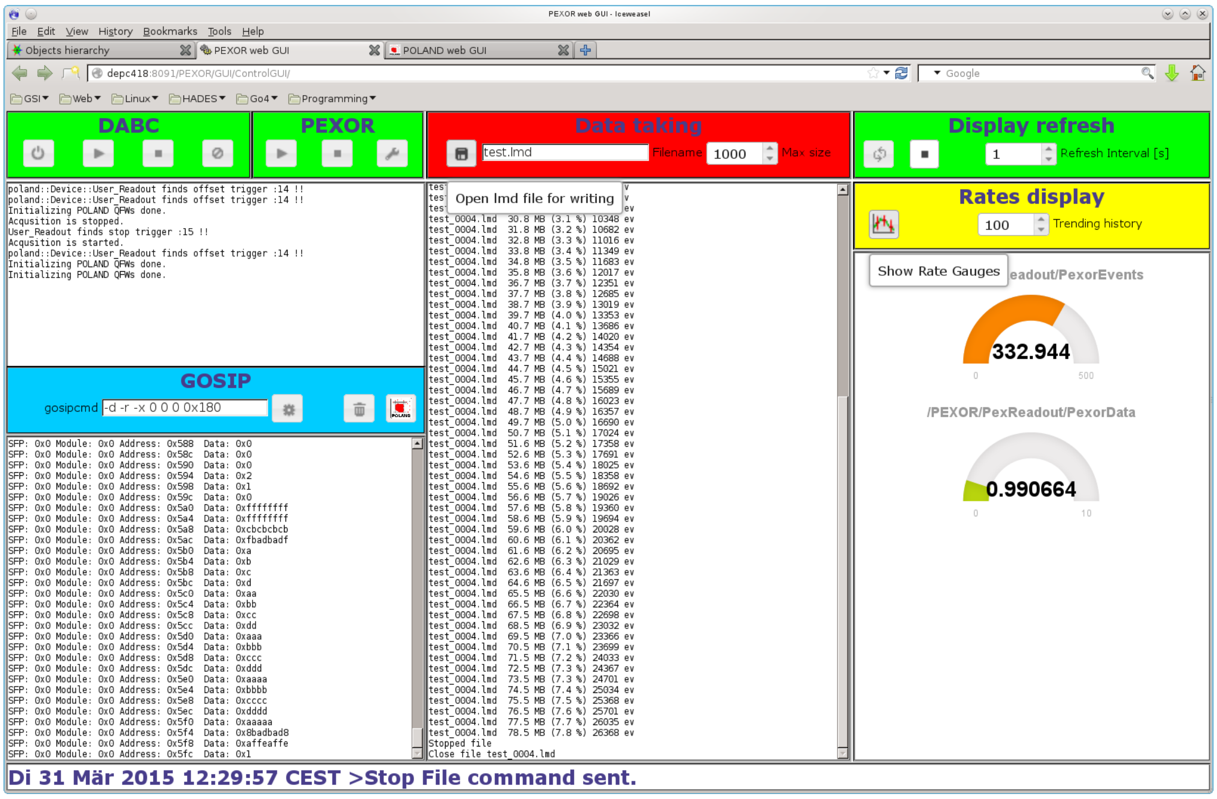


Figure 2. Heterogeneous distributed application control with DABC: The master process integrates object hierarchies of various agents and provides access at a single HTTP server. See text for details.

Figure 2 illustrates an example of a distributed set-up: the master process is linked via sockets of a specific plug-in to an MBS data acquisition node. This allows monitoring and control of such external DAQ system by means of the DABC web server. Additionally, another DABC HADES DAQ node and a Go4 analysis node are connected to the master via native DABC control sockets. The combined node hierarchy is delivered both at the master's HTTP server, and at a native DABC control socket. The latter may be applied for another level of hierarchy integration at a "super master".

The standard DABC web server delivers a JavaScript/jquery.ui GUI that offers to browse the entire object hierarchy in a tree view and display or even modify any object in its predefined way. Based on such a JavaScript API, any DABC plug-in may provide a more specialized browser GUI, tailored for the control requirements of the application. Extension hooks in the DABC web hierarchy allow to display this user defined GUI together with the standard GUI, i.e. clicking a special item in the hierarchy browser, or choosing the appropriate URL path as browser address, will display the specific GUI. Figure 3 contains a screenshot of a user defined browser GUI, as developed for the PEXOR/KINPEX plug-in. All relevant commands are represented as buttons. If clicked, an HTTP request will invoke the command at the web server. The control buttons are grouped in sections for generic DABC commands (configure, start, stop, shutdown), PEXOR specific commands (*StartAcquisition*, *StopAcquisition, InitDaq*), data taking commands (open/close file), and GOSIP commands for direct register access to the frontend electronics. DABC messages and command results are shown in text fields at the appropriate browser division. Additionally, event and data rates are drawn as gauge or trending graphs. Refresh frequency for monitoring such variables is adjustable in another control field.



Filename: test.lmd
Max size: 1000
Refresh Interval [s]: 1
Trending history: 100
gosipcmd: -d -r -x 0 0 0 0x180

```
poland::Device::User_Readout finds offset trigger :14 !!
poland::Device::User_Readout finds offset trigger :14 !!
Initializing POLAND QFWs done.
Acqusition is stopped.
User_Readout finds stop trigger :15 !!
Acqusition is started.
poland::Device::User_Readout finds offset trigger :14 !!
Initializing POLAND QFWs done.
Initializing POLAND QFWs done.
```

```
SFP: 0x0 Module: 0x0 Address: 0x588  Data: 0x0
SFP: 0x0 Module: 0x0 Address: 0x58c  Data: 0x0
SFP: 0x0 Module: 0x0 Address: 0x590  Data: 0x0
SFP: 0x0 Module: 0x0 Address: 0x594  Data: 0x2
SFP: 0x0 Module: 0x0 Address: 0x598  Data: 0x1
SFP: 0x0 Module: 0x0 Address: 0x59c  Data: 0x0
SFP: 0x0 Module: 0x0 Address: 0x5a0  Data: 0xffffffff
SFP: 0x0 Module: 0x0 Address: 0x5a4  Data: 0xffffffff
SFP: 0x0 Module: 0x0 Address: 0x5a8  Data: 0xcbcbcbcb
SFP: 0x0 Module: 0x0 Address: 0x5ac  Data: 0xfbadbadf
SFP: 0x0 Module: 0x0 Address: 0x5b0  Data: 0xa
SFP: 0x0 Module: 0x0 Address: 0x5b4  Data: 0xb
SFP: 0x0 Module: 0x0 Address: 0x5b8  Data: 0xc
SFP: 0x0 Module: 0x0 Address: 0x5bc  Data: 0xd
SFP: 0x0 Module: 0x0 Address: 0x5c0  Data: 0xaa
SFP: 0x0 Module: 0x0 Address: 0x5c4  Data: 0xbb
SFP: 0x0 Module: 0x0 Address: 0x5c8  Data: 0xcc
SFP: 0x0 Module: 0x0 Address: 0x5cc  Data: 0xdd
SFP: 0x0 Module: 0x0 Address: 0x5d0  Data: 0xaaa
SFP: 0x0 Module: 0x0 Address: 0x5d4  Data: 0xbbb
SFP: 0x0 Module: 0x0 Address: 0x5d8  Data: 0xccc
SFP: 0x0 Module: 0x0 Address: 0x5dc  Data: 0xddd
SFP: 0x0 Module: 0x0 Address: 0x5e0  Data: 0xaaaa
SFP: 0x0 Module: 0x0 Address: 0x5e4  Data: 0xbbbb
SFP: 0x0 Module: 0x0 Address: 0x5e8  Data: 0xcccc
SFP: 0x0 Module: 0x0 Address: 0x5ec  Data: 0xdddd
SFP: 0x0 Module: 0x0 Address: 0x5f0  Data: 0xaaaaa
SFP: 0x0 Module: 0x0 Address: 0x5f4  Data: 0x8badbad8
SFP: 0x0 Module: 0x0 Address: 0x5f8  Data: 0xaffeaffe
SFP: 0x0 Module: 0x0 Address: 0x5fc  Data: 0x1
```

```
tes                                   v
tes                                   v
tes                                   ev
test_0004.lmd  30.8 MB (3.1 %) 10348 ev
test_0004.lmd  31.8 MB (3.2 %) 10682 ev
test_0004.lmd  32.8 MB (3.3 %) 11016 ev
test_0004.lmd  33.8 MB (3.4 %) 11349 ev
test_0004.lmd  34.8 MB (3.5 %) 11683 ev
test_0004.lmd  35.8 MB (3.6 %) 12017 ev
test_0004.lmd  36.7 MB (3.7 %) 12351 ev
test_0004.lmd  37.7 MB (3.8 %) 12685 ev
test_0004.lmd  38.7 MB (3.9 %) 13019 ev
test_0004.lmd  39.7 MB (4.0 %) 13353 ev
test_0004.lmd  40.7 MB (4.1 %) 13686 ev
test_0004.lmd  41.7 MB (4.2 %) 14020 ev
test_0004.lmd  42.7 MB (4.3 %) 14354 ev
test_0004.lmd  43.7 MB (4.4 %) 14688 ev
test_0004.lmd  44.7 MB (4.5 %) 15021 ev
test_0004.lmd  45.7 MB (4.6 %) 15355 ev
test_0004.lmd  46.7 MB (4.7 %) 15689 ev
test_0004.lmd  47.7 MB (4.8 %) 16023 ev
test_0004.lmd  48.7 MB (4.9 %) 16357 ev
test_0004.lmd  49.7 MB (5.0 %) 16690 ev
test_0004.lmd  50.7 MB (5.1 %) 17024 ev
test_0004.lmd  51.6 MB (5.2 %) 17358 ev
test_0004.lmd  52.6 MB (5.3 %) 17691 ev
test_0004.lmd  53.6 MB (5.4 %) 18025 ev
test_0004.lmd  54.6 MB (5.5 %) 18358 ev
test_0004.lmd  55.6 MB (5.6 %) 18692 ev
test_0004.lmd  56.6 MB (5.7 %) 19026 ev
test_0004.lmd  57.6 MB (5.8 %) 19360 ev
test_0004.lmd  58.6 MB (5.9 %) 19694 ev
test_0004.lmd  59.6 MB (6.0 %) 20028 ev
test_0004.lmd  60.6 MB (6.1 %) 20362 ev
test_0004.lmd  61.6 MB (6.2 %) 20695 ev
test_0004.lmd  62.6 MB (6.3 %) 21029 ev
test_0004.lmd  63.6 MB (6.4 %) 21363 ev
test_0004.lmd  64.6 MB (6.5 %) 21697 ev
test_0004.lmd  65.5 MB (6.6 %) 22030 ev
test_0004.lmd  66.5 MB (6.7 %) 22364 ev
test_0004.lmd  67.5 MB (6.8 %) 22698 ev
test_0004.lmd  68.5 MB (6.9 %) 23032 ev
test_0004.lmd  69.5 MB (7.0 %) 23366 ev
test_0004.lmd  70.5 MB (7.1 %) 23699 ev
test_0004.lmd  71.5 MB (7.2 %) 24033 ev
test_0004.lmd  72.5 MB (7.3 %) 24367 ev
test_0004.lmd  73.5 MB (7.3 %) 24701 ev
test_0004.lmd  74.5 MB (7.4 %) 25034 ev
test_0004.lmd  75.5 MB (7.5 %) 25368 ev
test_0004.lmd  76.5 MB (7.6 %) 25701 ev
test_0004.lmd  77.5 MB (7.7 %) 26035 ev
test_0004.lmd  78.5 MB (7.8 %) 26368 ev
Stopped file
Close file test_0004.lmd
```

Di 31 Mär 2015 12:29:57 CEST >Stop File command sent.

Figure 3. Example of a user-defined browser GUI in DABC as written for the PEXOR plug-in. See text for details.

4. Application for HADES experiment

The HADES experiment has one of the largest detector set-ups at GSI with about 30 sub-detector units [6]. Their data are read out via Gbit Ethernet connections with the *trbnet* format, then combined and stored by means of 4 Linux-based event builder computers.

For several test set-ups, a DABC plug-in had already been developed that could receive and combine such *trbnet* data packets via UDP connections [7]. These developments have been completed to cover the full functionality of the previous HADES event building software. This includes interfacing the EPICS based HADES control system, the ORACLE run statistic data base, and the RFIO tape storage of GSI. Initialization procedures of the DABC event builder processes have been fully integrated into the existing HADES DAQ configuration. Moreover, by means of DABC stream server socket the connection to HADES online quality monitoring analysis could be improved.

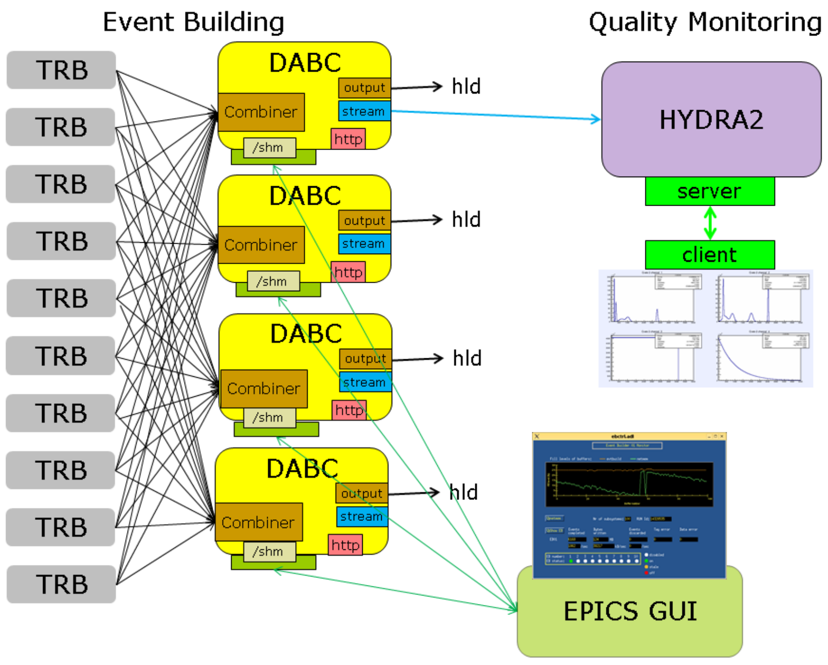


Figure 4. HADES data acquisition and quality monitoring with DABC event builders.

Figure 4 shows the HADES data acquisition and online monitoring components with DABC. About 30 TRB front-end hubs send sub-event data to each of 16 DABC event builder processes in a round robin scheme. Each event builder writes full event data in *hld* file format both to an array of local hard disks, and via Gbit Ethernet sockets to the RFIO tape library of GSI. Additionally, one event builder provides a sample of the event data for a HYDRA2 quality analysis (QA) with a live monitoring display. The previous *hadaq* event builders had realized such samples by writing temporary *hld* files to be read back by the QA process via an NFS mounted hard disk. With DABC, event data samples are now delivered to the QA client via an “MBS stream server” TCP/IP socket. Several different analysis clients could connect to this server and can receive the same event samples simultaneously. The existing EPICS control GUI and run synchronisation system has been interfaced via shared memory to DABC in the same way as the previous event building software *hadaq*. Moreover, the existing start-up scripts have been extended in a way that each event builder process can run either DABC or *hadaq* with the same settings. So it is possible to operate the DAQ with “mixed” event builders (DABC and *hadaq*). Furthermore, DABC log output has been attached to the HADES *syslogd* -based logging facility. A data taking “run statistics” interface to an ORACLE data base has also been implemented in DABC, alike in *hadaq* event builders, by means of intermediate text files.

In 2014 DABC was applied for data taking of beam times at HADES experiment. With a proton beam in May 2014 and a pion beam in July 2014, for the first time one of 16 event builder processes ran with DABC, all others still with previous *hadaq* software. Finally with a pion beam in September 2014, DABC was running all event builders. Here 29 frontend systems usually fed 3 active event builder processes at typical pion beam event rates of < 2 kHz average. The performance limit of this system with full disk and tape file output has been measured with a 25 kHz pulsed trigger. In this test each event builder could handle data rates of 38Mbyte/s without data loss at the UDP transport layer.

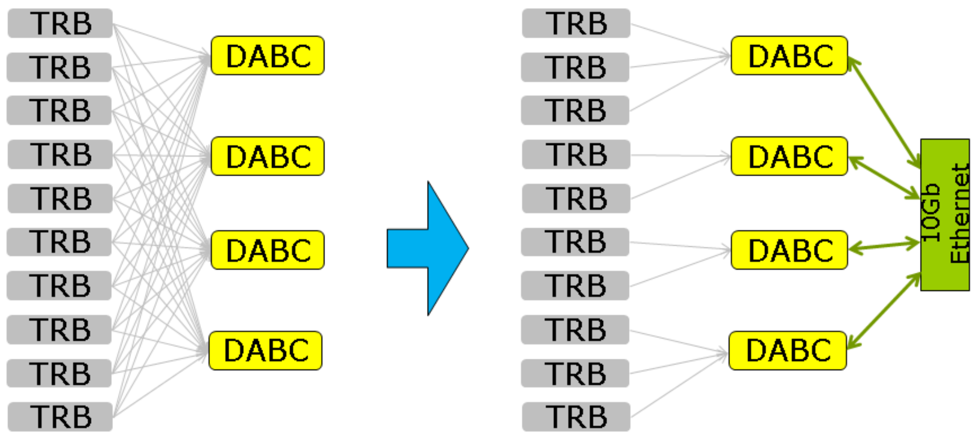


Figure 5. HADES event building topology: exisiting (left) and planned (right).

The next HADES DAQ upgrade is going to integrate numerous new TRB3 front-end boards with partially “free running” readout and FPGA-TDC features for RICH and ECAL sub-detectors. In the scope of such upgrade it is also planned to change the topology of the event building network. Instead of the existing “n x m” connection between 30 sub-event hubs and 16 event builder nodes with UDP connections (Figure 5, left), it might be preferable to configure dedicated sub-event receiver nodes for each frontend sender (Figure 5, right). So sub-event specific data processing, e.g. TDC online calibration, can be applied directly at the very receiver node. The actual event building, i.e. combination of all corresponding sub-events, is then performed in a subsequent “builder network” (B-net) between such sub-event receivers and the event builder processes. This B-net can use lossless TCP/IP or InfiniBand connections instead of UDP and may be better tuned with respect to data traffic shaping for high bandwidth performance. Since DABC was designed from the beginning to optimize such event builder networks [8], it is well suited to implement such topology for HADES. Tests on an 800 nodes QDR InfiniBand cluster (LOEWE CSC, Frankfurt, Germany) in 2011 have shown that DABC could achieve with this B-net in “all to all” traffic pattern about 80% of available bandwidth, i.e.1.25 TB/s.

5. Conclusions and outlook

The DABC framework has been advanced to version 2 with several improvements. The new web server features, together with the possibility to merge hierarchies of remote agents at a single master gateway, promise to monitor and control even large distributed data acquisition or analysis nodes at any web browser. The DABC integration of standard GSI readout hardware has been further developed. For PEXOR/KINPEX systems this already gives a working alternative to the established DAQ system MBS. However, a full replacement of MBS by DABC is not intended and not easily possible, because of the large variety of readout hardware and platforms supported by MBS. On the contrary, it is provided that any existing MBS node can be controlled as agent from a DABC master web server with a dedicated browser GUI.

The HADES experiment has replaced its old event building software by DABC and applied it for production data taking at the pion beam campaign in 2014. DABC could handle all requirements and facilitated data connections to quality analysis monitoring. To fully utilize DABC possibilities, HADES will soon test another event building topology with a lossless DABC builder network (Bnet). Additionally, the HADES monitoring system may also apply the DABC web server and control hierarchies in the future.

References

[1] Adamczewski-Musch J, Essel H G, Kurz N and Linev S 2010 Dataflow engine in daq backbone dabc *IEEE TNS* **57** 614-17

[2] Minami S, Hoffmann J, Kurz N and Ott W 2010 Design and implementation of a data transfer protocol via optical fibre, *Proc. RT 2010 17th IEEE-NPSS (Lisbon),* http://dx.doi.org/10.1109/RTC.2010.5750447

[3] Adamczewski-Musch J, Essel H G and Linev S 2011 The dabc framework interface to readout hardware *IEEE TNS* **58** 1728-32

[4] Essel H G, Hoffmann J, Kurz N and Ott W 2000 The general purpose data acquisition system mbs, *IEEE TNS* **47** 337-9

[5] Adamczewski-Musch J, Bellenot B and Linev S 2014 Web interface for online root and daq applications, *Proc. RT2014 19th IEEE-NPSS (Nara),* http://dx.doi.org/10.1109/RTC.2014.7097456

[6] Michel J et al. 2011, The upgraded hades trigger and data acquisition system JINST **6** C12056

[7] Adamczewski-Musch J, Linev D, Ovcharenko E and Ugur C, 2013 HADES trbnet data formats for dabc and go4 *GSI Scientific Report* **GSI-SR2012** PHN-SIS18-ACC-41

[8] Adamczewski J, Essel H G, Kurz N and Linev S 2008 Data acquisition backbone core dabc *IEEE TNS* **55 No.1**, 251-5