FAIR Facility for Antiproton and Ion Research

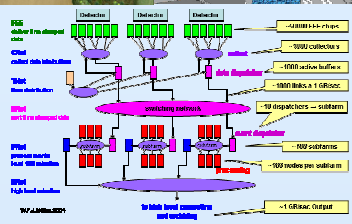
GSI

http://dabc.gsi.de

Data Acquisition Backbone Core DABC

J. Adamczewski-Musch, H.G. Essel, N. Kurz, S. Linev

GSI Helmholtzzentrum für Schwerionenforschung GmbH, Darmstadt



Motivation for developing DABC

Use cases

- Detector tests
- FE equipment tests
- High speed data transport
- Time distribution
- Switched event building
- Software evaluation
- MBS* event builder

Requirements

- build events over fast networks
- handle triggered or self-triggered front-ends
- process time stamped data streams
- provide data flow control (to front-ends)
- connect (nearly) any front-ends
- provide interfaces to plug in application codes
- connect MBS readout or collector nodes
- be controllable by several controls frameworks

General purpose DAQ, final features

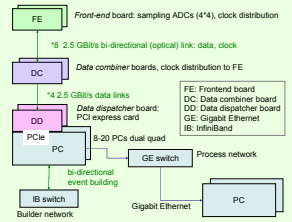
- Event building network with standard components like InfiniBand
- Scheduled data transfer with ~10µs accuracy
- Thread synchronization with determined latency
- Needs realtime features of standard Linux kernel since 2.6.22:
  - RT priorities, nanosleep and pthread_condition
  - PREEMPT_RT, high resolution timer

Using the real time features of Linux

- Linux kernel 2.6.22: high resolution timer.
- With two threads doing condition ping-pong and one "worker"-thread one turn is 3.2 µs
- With high resolution timer and hardware priority a sleeping thread gets control after 10 µs.
- Synchronized transfer with microsleep achieves **800 MB/sec**

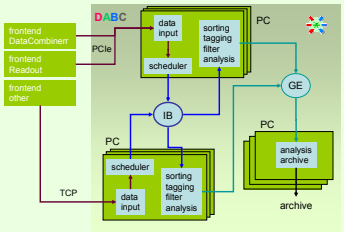
* Muliti Branch System: current standard DAQ at GSI

Frontend hardware example

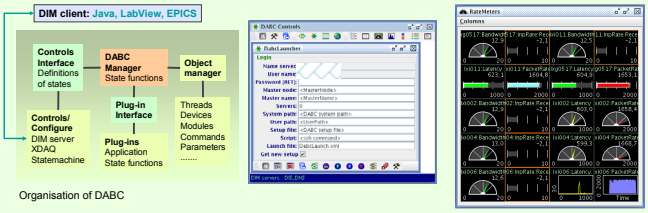


The **left figure** shows hardware components the DABC has to deal with. Front-end boards typically keep sampling digitizers, whereas combiner and dispatcher boards have fast data links controlled by FPGAs. Prototypes of these boards are under test. Data rates in the order of 1 GBytes/s could be delivered through the PCI express interface.

Logically such a setup looks like sketched in **right figure**. Depending on the performance requirements (data rates of the front-end channels) one may connect one or more front-ends to one DABC node. From a minimum of one PC with Ethernet up to medium sized clusters all kind of configurations are possible. One may separate input and processing nodes or combine both tasks on each machine using bi-directional links depending on CPU requirements.

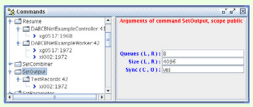


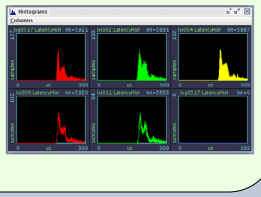
DABC controls (Java & DIM)



Organisation of DABC

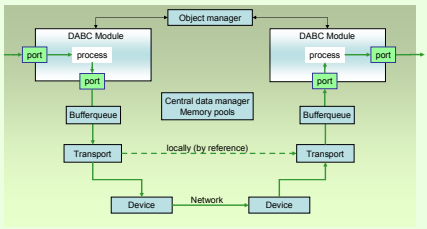
The current Java GUI is built up dynamically on the information delivered by the DIM servers running in the application threads on all nodes

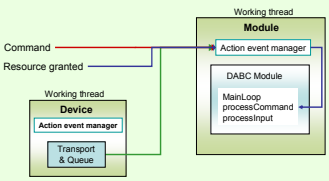




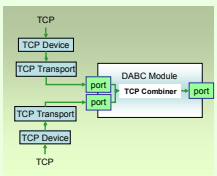
DABC as data flow engine

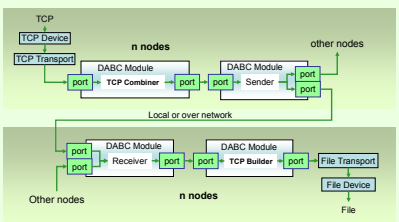
The **right figure** shows the components of DABC data flow mechanism. Basic object is a *Module* which gets access to data buffer queues through input *ports*, processes them, and outputs them to output *ports*. The data queues on each node are controlled by a *Memory management* which allows processing buffers without copy by modules running on the same node. The *Transport* component in this case propagates only references. If a *port* is connected to a remote *port*, the *Transport* and *Device* components do the transfers.





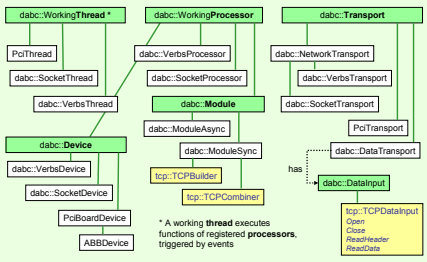
Module and *Device* components can either run in separate threads (**left figure**) to utilize multiple CPUs, or in one thread which is faster because no module synchronization is needed in this case. Data processing functions of the module are called by an *Action event manager*. The data flow is controlled by buffer resources (queues) handled through the *Transports*. Once a queue buffer is filled, the *Transport* sends a signal to the *Action event manager*, which in turn calls the *processInput* function of the module associated through a port. This function has access to the data buffers through the port (**figure above**). Similarly, commands can be sent to the *Action event manager*, which calls the associated module function *processCommand*.





Two examples of data flow configurations for TCP front-ends like MBS.
Left graph shows event building by combiner module on one event builder node.
Right graph shows n nodes with one input channel each, n event builders and event building over network

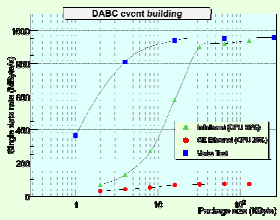
Class hierarchy and plug-ins



* A working **thread** executes functions of registered **processors**, triggered by events

Layers of main DABC data flow classes. Base classes in green. Application plug-in classes in yellow for TCP front-ends like MBS. These have to be implemented for the examples shown in the right middle panel *DABC as data flow engine*.

Performance measurements InfiniBand



Event building
Bandwidth achieved by DABC event building on 4 nodes at GSI (non synchronized mode, **left graph**). The test measurement result is included for comparison. With buffers > 32KB DABC delivers nearly the bandwidth of the plain test program (**950 MBytes/s**). Gigabit Ethernet delivers **70MByte/s**. Four data sources per node generate data. Event building on the receiver nodes only checks the completeness of data.

Work supported by EU RP6 project JRA1 FutureDAQ RII3-CT-2004-506078