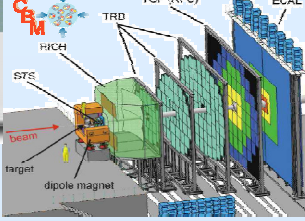
FAIR Facility for Antiproton and Ion Research

GSI



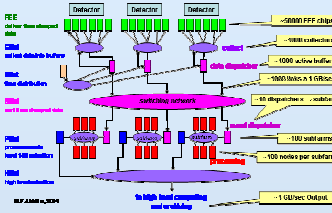
Compressed Baryonic Matter experiment at FAIR

Data Acquisition Backbone Core DABC

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

GSI, Darmstadt

The new Facility for Antiproton and Ion Research at GSI



First sketch of the data acquisition concept for CBM (2004)
H.G.Essel, *FutureDAQ for CBM: On-line event selection*,
IEEE TNS Vol.53, No.3, June 2006, pp 677-681

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

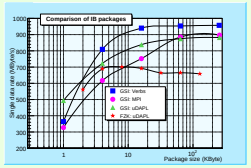
Requirements
- build events over fast networks
- handle triggered or self-triggered front-ends
- process time stamped data streams
- provide data flow control (to front-ends)
- connect (nearly) any front-ends
- provide interfaces to plug in application codes
- connect MBS readout or collector nodes
-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

General purpose DAQ
- Event building network with standard components like InfiniBand
- Scheduled data transfer with ~10µs accuracy
- Thread synchronization with determined latency

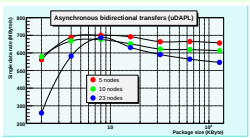
Needs realtime patches of standard Linux kernel 2.6.19:
- RT priorities, nanosleep and pthread_condition
- PREEMPT_RT (Ingo Molnar), high resolution timer (Thomas Gleixner)

* Muliti Branch System: current standard DAQ at GSI

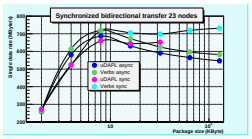
Performance measurements InfiniBand



All nodes send to all nodes (bi-directional). Data rates are per direction. In synchronized mode senders send data according a time schedule avoiding conflicts at the receivers. Without synchronization all senders send round robin to all receivers and the network must handle collisions.
Measurements have been performed at GSI (4 nodes, graph on top) and Forschungszentrum Karlsruhe (IWF) on a cluster with 23 double dual-core Opteron (2.2 GHz)* and SilverStorm (QLogic) switch. The GSI nodes are considerably faster than the FZK nodes.



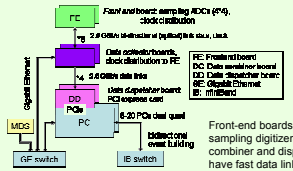
Scaling from 5 to 23 nodes to 85% which is 50% nominal bandwidth (non synchronized traffic).
With 8 KB buffers traffic can be best optimized by switch. With larger buffers the scaling drops.



Effect of scheduling on 23 nodes.
Scheduled transfer is better than asynchronous above 64 KB buffer size.
Verbs is better than uDAPL as already shown in the top graph.
With larger buffers the synchronized transfer is 25% higher.

* Many thanks to Frank Schmitz and Ivan Kondov

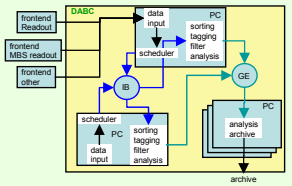
Frontend hardware example



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.

DABC as data flow engine

Logical structure of DABC



Logically such a setup looks like sketched in **Figure left**. 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 bidirectional links depending on CPU requirements.

Application plug-ins to handle the data

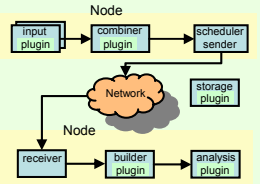
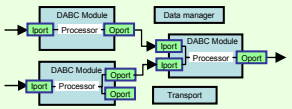


Figure right shows a typical setup of components and where application plug-ins are needed. On the input nodes run threads to get data (input), combine/transform/check them (combiner thread), and send them over network to build nodes (scheduler/sender threads), where they are received (receiver threads), events are built (builder thread) and analyzed (analysis thread).

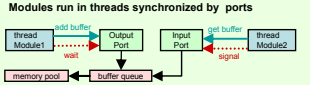
Data flow through modules connected by ports



The central component is a *module* (processing thread) which gets access to data buffers through IO ports (**Figure left**).

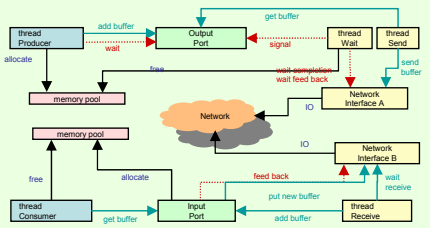
Data flow control of DABC

Modules run in threads synchronized by ports



The communication between the module threads (Figure left) is done by thread conditions. The data flow is controlled by buffer resources (queues).

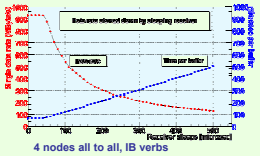
Back pressure over network



The special case is here that the last output port leads to the network.
If the receiver nodes are busy with other calculations like event building, it is necessary to hold the senders. This is achieved by the back pressure mechanism shown in Figure above. The receivers inform the senders about their queue status after a collection of buffers. The senders only send when the receivers signalled sufficient resources. The back pressure works smoothly.

Real time performance

- GSI: Cluster with 4 double Opteron machines (2.2 GHz), and Mellanox switch. Linux kernel 2.6.19 with RT patches (high resolution timer now in 2.6.21). InfiniBand OFED 1.2.pre verbs linbrary.
- 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**



In the figure on the left the receiver threads increase a sleep time from zero to 500 µs. One can see that the transfer time per buffer is exactly the sleeping time with a minimum of 70 µs. That means that no overhead is introduced by the feed back messages.

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