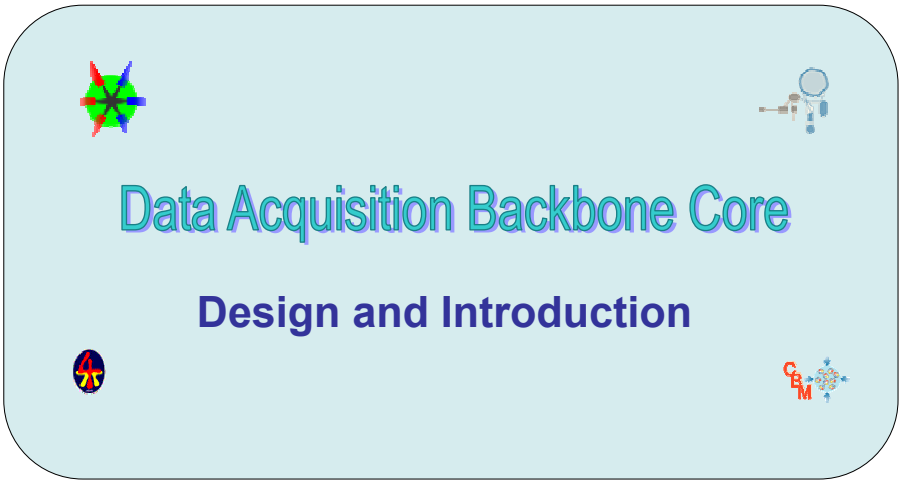

Document	Date	Editor	Revision	Comment
DAS-06	2006 Mar 13	Hans G.Essel	0.9	First scetch

March 2006

GSI Darmstadt

Contents

Preface

This document describes the requirements, design, and implementation of the general purpose data acquisition demonstrator system. This system, called *Demonstrator*, is a result of the discussions about DAQ concepts for CBM, Panda, and FutureDAQ started in 2004.

0.1 Structure of document

The document is structured hierarchically. To make sure that files to be included by \input{filename} or \include{filename} can be located, set the following environment variables:
Linux:
export TEXINPUTS=<topdirectory>//:
Windows: If one uses fpTeX with WInEdt:
Append ;P:\Application\TeXLive2005\bin\win32 to PATH.
Set TEXINPUTS to x:\topdirectory\//;
(Systemsteuerung->System:Erweitert:Umgebungsvariablen)

The full document is built by command (we are on topdirectory):

```
pdflatex main-all
makeindex
pdflatex main-all
```

On each subdirectory there is a main file main-xxx.tex, i.e.

```
cd template
pdflatex main-XXX
makeindex
pdflatex main-XXX
```

The files on directory `template` can be used as templates, i.e. copied to a new subdirectory. All occurences of XXX in file names and tex files should then be renamed properly. The script `rename.sh` can be used to do so:

```
. ./rename.sh XXX yyy
```

replaces all `XXX` to `yyy` in tex file names and tex files. (After that all `*XXX*` files can be deletetd).
The file `XXX-section.tex` contains commonly used tex commands. It could be used as cut&paste source.
Description of the files:

0.1.0.1 Topdirectory

main-all.tex main file to be texed. Includes all steer files from subdirectories.
bibitem.tex references
demo-glossary.tex glossary
demo-requirements.tex brief and informal list of requirements
democlass.cls document description

0.1.0.2 Subdirectory `environment`

demo-docrev.tex document name and revision information
demo-defs.tex central definitions (included by all main files)
demo-post.tex reference and index chapters (included by all main files)
demo-frontpage.tex first page of top document
demo-people.tex list of people
demo-preface.tex this text
demo-work.tex working packages

0.1.0.3 Subdirectory `controls`

ctrl-docrev.tex document name and revision information. Is included by `main-all.tex` and `main-controls.tex`
main-controls.tex main file to be texed. Includes `steer-controls.tex` and `ctrl-docrev.tex`.
Adjust document information here.
steer-controls.tex includes everything needed from this directory. Is included by `main-all.tex`

All other directories below topdirectory have the main, docrev and the steer file.

0.2 Naming conventions

The Demonstrator Collaboration

- **Budapest, Hungary, Eötvös University** [1] F. Deak, R. Izsak, A. Kiss
- **Budapest, Hungary, KFKI** [1] E. Denes, Z. Fodor, J. Kecskemeti, Cs. Soos, T. Kiss, G. Vesztergombi
- **Darmstadt, Germany, GSI** J. Adamczewski, E. Badura, H. Deppe, H. Essel, H. Flemming, B. Kolb, S. Linev, W.F.J. Müller
- **Heidelberg, Germany, Kirchhoff-Institut für Physik, Universität Heidelberg**[1] M. Alcocer, D. Atanasov, U. Kebschull (Universität Leipzig), I. Kisel, V. Lindenstruth, G. Torralba, G. Tröger
- **Kaiserslautern, Deutschland, Universität Kaiserslautern** [1] D. Muthers, R. Tielert, S. Tontisirin
- **Mannheim, Germany, Inst. of Computer Engineering, Universität Mannheim** [1] K.-H. Brenner, U. Brüning, P. Fischer, H. Fröning, J. Gläß, P. Haspel, A. Kugel, R. Männer, D. Slogsnat, C. Steinle, D. Wohlfeld, A. Wurz

Acknowledgement

We acknowledge the support of the European Community-Research Infrastructure Activity under the FP6 "Structuring the European Research Area" programme (HadronPhysics, contract number RII3-CT-2004-506078). Besides GSI, universities of Heidelberg, Mannheim, Munich, Katowice, Krakow, Warsow, Giessen, Budapest, and Turino are participating.

[1] membership to be confirmed

Requirements pin board

This list is rather about reminders. As soon as the items are addressed somewhere else, they can be eliminated here.

1. It must be possible to partition the DAQ by configuration.
2. System must run with or without trigger

1 Documents

Document names: DAS-t-c-d, where t=R(equirement) or D(esign), c=component key, d=year.

Titel: FutureDAQ Demonstrator Introduction

Document	Date	Editor	Revision	Comment
DAS-D-IN-06	2006 Oct 20	Hans G.Essel	0.9-2	First scetch

Titel: FutureDAQ Demonstrator

Document	Date	Editor	Revision	Comment
DAS-D-dtor-06	2006 Oct 23	Hans G.Essel	0.9	First scetch

Titel: FutureDAQ Demonstrator Simulations

Document	Date	Editor	Revision	Comment
DAS-SIMU-06	2006 Mar 13	Hans G.Essel	0.9	First scetch

Titel: FutureDAQ Demonstrator Testing

Document	Date	Autor	Revision	Comment
DAS-TEST-06	2006 Nov 1	J.Adamczewski S.Linev	0.9	Draft

Titel: FutureDAQ Demonstrator Software

Document	Date	Editor	Revision	Comment
DAS-D-SW-06	2006 Oct 20	Hans G.Essel	0.9-2	First scetch

Titel: FutureDAQ Demonstrator Hardware

Document	Date	Editor	Revision	Comment
DAS-D-HW-06	2006 Mar 13	Hans G.Essel	0.9	First scetch

Titel: FutureDAQ Demonstrator Controls

Document	Date	Editor	Revision	Comment
DAS-D-CTRL-06	2006 Mar 13	Hans G.Essel	0.9	First scetch

2 Introduction

2.1 DAQ for CBM

The communication and processing needed between the front-end electronics, generating digitized detector information, and the archival storage, where the complete context of selected candidate events is recorded, can be structured and organized in several ways. The solution described in [1] is guided by two principles: processing is done after event building and it is done in a structured processor farm. It is well adapted to the type of processing needed in the CBM experiment and leads to a straightforward and modular architecture.

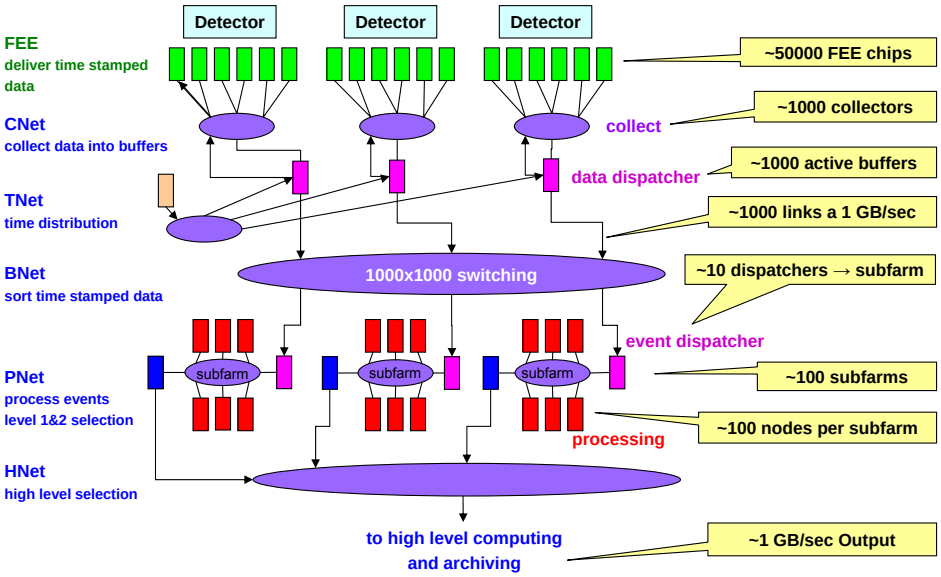

Figure 2.1: CBM overall data processing architecture

A logical data flow diagram is shown in Fig. 2.1, indicating the data sources and processing elements as boxes and every form of interconnection networks as ovals.

2.2 Demonstrator mission

[Marker:in.mission]
The final DAQ system will be quite complex and involves many technologies which are at the limits of today possibilities. Therefore it is necessary to start early with a system which can be implemented in the next years without showing up the full final performance needed. This system, a *Demonstrator*, is useful in four respects.

2.2.1 Demonstration of key technologies

The first task is to prove the key technologies needed, maybe with less performance.

- FEE: self-triggered, data push, conditional RoI based readout
- CNet: combined data, time, control, and RoI traffic
- TNet: low jitter clock and synchronization over serial links
- BNet: high bandwidth switched network for event building
- ECS: integrated control system

2.2.2 Test bed for prototypes

All components will need iterations. A testing environment is needed from the beginning.

- Hardware
- Firmware
- Controls
- Software

2.2.3 Functional DAQ for detector tests

There are many questions concerning detector performance. Prototypes will be developed. They can only be tested with a DAQ system with the same characteristics as the final one. The whole readout chain must be available to prove that a triggerless readout is feasible.

2.2.4 General purpose DAQ for medium sized experiments

To prove that a highly distributed system scales it is necessary to install at least a medium sized setup. This can only be afforded if it is used in production for an experiment.

2.3 Demonstrator use cases

According the mission there are several use cases or scenarios:

2.3.0.1 Frontend test bed

The new components FEB, DCB, and ABB and their connections must be tested. The timing system and the triggered/non-triggered mode must be tested.

2.3.0.2 Detector test bed

Detector tests with a free running system are needed to verify that the data rates are not very much bigger that expected (dark noise).

2.3.0.3 Switched event building

Independent of the data sources (ABB or GEB) several mechanisms for the switched event building must be implemented and tested.

2.3.0.4 DAQ framework and control systems

There are still several options for the choice of the control system.

2.3.0.5 MBS replacement

Very probably there will be no complete replacement of the MBS possible for the next years. One could envision that the MBS frontend nodes, i.e. the readout nodes send their subevent data peer to peer to new event builder nodes (new MBS mode).

2.3.0.6 Hybrid setup

If the system should run in production, very probably a hybrid of MBS for ancillary components and new frontends will be needed.

2.4 Demonstrator architecture

As a first step the *Demonstrator* shall be used to investigate practically the technologies needed.

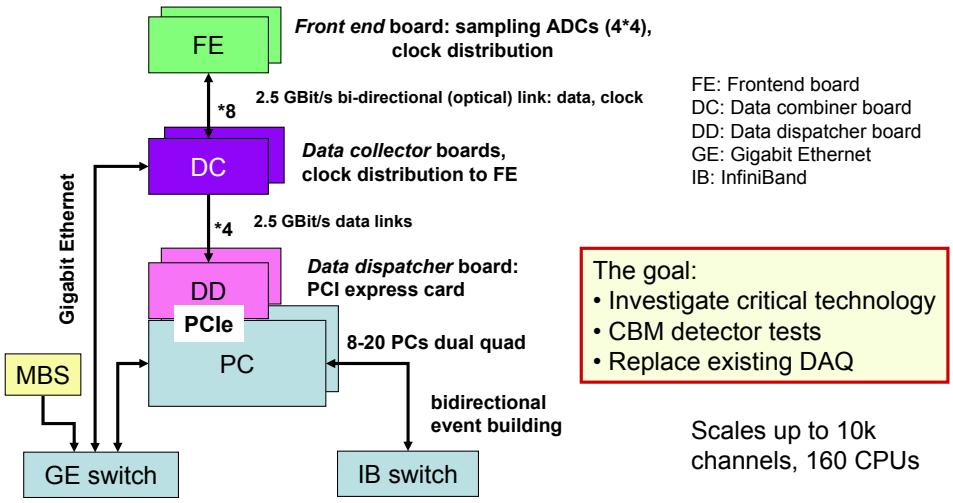

Figure 2.2: *Demonstrator* overall data processing architecture

The system can be described in several dimensions:

Hardware There are mainly three boards with different tasks but similar architecture.
Data formats The data and time stamp formats must be defined early because they are interpreted at many occasions. A change would have big impact.
Data flow models Mechanisms how the data is moving throughout the system.
Software framework What frameworks are used. We should not start from scratch.
High level software Runs wherever Linux is running.
Board level software Mainly the FPGA codes
Controls Several sorts of controls

2.4.1 Hardware

The hardware components as shown in Fig. 2.2 are summarized with their key features.

2.4.1.1 Front End Electronics board FEE

General purpose chamber-mountable board with ADCs (4*4 channels, 65 MHz sampling, 10-12 bit), FPGA (Virtex-4, PPC, Ethernet MAC, MGT plus SDRAM, CPLD, NVRAM), plugable to preamp/chapers. A version utilizing pipeline TDCs (ns resolution) possible. Two clock domains: receive clock of 312.5 MHz (timing and trigger) and sampling clock with 62.5 MHz (measurement). Four pair of LVDS bi-directional links (625 Mbps).

2.4.1.2 Data Combiner board DCB

Four bi-directional LVDS links to FEB, Ethernet, MGT with SFP optical link to ABB, same FPGA. A test layout of such board is shown in Fig. 4.1. A possible connection to the between FEE and DCB boards is shown if Fig. 4.2

2.4.1.3 Active Buffer board ABB

PCIe board, 4 - 8 MGT/SFP optical links to DCBs, same FPGA.

2.4.1.4 Timing board

Similar to DCB, gets optional trigger inputs, generates clock and distributes through optical splitters to DCBs.

2.4.1.5 Event builder

Standard PC with GE, InifiniBand switch.

2.4.2 Data formats

See also Software paper.

2.4.2.1 Raw data stream

The data must be formatted on the FEE boards to achieve the following requirements:

1. compressed
2. coded geographical address
3. incremental time stamps
4. time epoch markers

The event building is done through a switched network. Logical entities to be switched are epochs. Data before switching may be stored/retrieved in a binary format for testing and/or development.. It should be foreseen that data has to be reformatted before the switching, because different subsystems might produce different data formats. Eventually the time sorting might be done. After the switching, the event definition has to be done. At that point all data must be repacked.

2.4.2.2 Event definition data

This intermediate format is a full time ordered compressed data stream of an epoch. Optionally it may contain multiplicity histograms to accelerate later event definition. Data may be stored/retrieved in this format.

2.4.2.3 Event format

After event definition events must be formatted and stored.

2.4.3 Software

2.4.3.1 Board level software

Software running on the FPGAs. The data flow from the FEE through the DCB to the ABB is controlled by FPGAs.

2.4.3.2 Data mover

The data dispatchers (task on standard PCs with Linux) read the data stream via PCIe from the ABB boards and send it via a fast network (GE or IB) switch to the event builder tasks (same PCs) using the links bi-directional.

2.4.3.3 Data processing

From the event builder task the formatted events are handed over to data processing tasks for event filtering and archiving. These tasks can run optionally on separate machines.

2.4.4 Controls

For the three main fields of detector controls, DAQ controls, and board controls there are currently three products under test:

EPICS Well known control system, widely used in accelarator and classical controls. SNMP interface is available for monitoring clusters and net devices.

LabView Standard slow control at GSI.

SysMES Configuration control system developed at KIP. Has CA interface to work with EPICS and SNMP.

xDAQ DAQ framework of CMS

Because it might be not possible to make a decision for one of these to be used exclusively one should investigate/implement interoperability. Communication standards in question:

- DIM: DIM server in an EPICS IOC as first test bridge
- CA: LabView CA client available
- SOAP: provided by xDAQ, Java interface available.
- CA: EPICS IOC (xDAQ application) gateway to xDAQ infospace

The DAQ controls must provide the following functionality:

- Control tasks on remote machines
- Communicate with all tasks
- Mechanism to store/retrieve the whole setup
- Monitor setup, status, data flow, and performance
- Control data flows
- Visualization and GUI

3 Demonstrator

3.1 Concept

[Marker:dtor.concept]
As mentioned in the introduction, there are several scenarios or use cases for the *Demonstrator* reflecting differing requirements and delivery dates.

3.1.1 *Demonstrator* use cases

3.1.1.1 Detector test bed

Detector tests with a free running system are needed to verify that the data rates are not very much bigger than expected (dark noise).

3.1.1.2 Switched event building

Independent of the data sources (*Active Buffer Board* or GigabitEthernet) several mechanisms for the switched event building must be implemented and tested.

3.1.1.3 DAQ framework and control systems

There are still several options for the choice of the control system. The χ*DAQ* framework provides several communication mechanisms. The control system not necessarily must be χ*DAQ* but could be e.g. EPICS based. In such a case gateways must be implemented.

3.1.1.4 *MBS* replacement

Very probably there will be no complete replacement of the *MBS* possible for the next years. A very big amount of existing hardware of running *MBS* systems cannot be replaced neither they can be attached directly to the new DAQ framework. At least the effort to replace the software running on the front-end CPUs (RIO, Lynx, *MBS*) would be too big, if possible. Instead one could envision that the *MBS* front-end nodes, i.e. the readout nodes, send their sub-event data peer to peer to new event builder nodes (new *MBS* mode). Each of these nodes functions as sub-event receiver (Ethernet, TCP) and sends the data for event building through the data transport network like InfiniBand to the others and functions also as event processor. The Ethernet input channel replaces the *Active Buffer Board* input channel.

3.1.1.5 Hybrid setup

If the system should run in production, very probably a hybrid of *MBS* for ancillary components and new front ends will be needed. When running in triggered mode, the trigger of the new components must be synchronized with the *MBS* trigger system. In triggerless mode the *MBS* data must be time stamped

or *MBS* triggers must be injected and recorded in one of the time stamped data streams by connecting the trigger bus directly to an *Front-end Electronics Board*.

3.1.1.6 Front end test bed

The new components *Front-end Electronics Board*, *Data Combiner Board*, and *Active Buffer Board* and their connections must be tested. The timing system and the triggered/non-triggered mode must be tested.

3.1.2 The name

Because the systems described might be also used by other experiments it might evolve to more than a demonstrator. At the other hand it will not provide components of the front-end side but rather provide the hooks and plug-ins to attach a big variety of front-end systems and components. There is one principal restriction, however, in the current design. There is nothing like a 2nd level trigger. That means, that we assume that the full data rate can be switched through a network for event building. However, a congestion prevention must be provided generating dead time for the detector readout. From this point on we have only event filters. The front-ends might however be triggered or free running.

The demonstrator as a production system would be rather a backbone. Therefore we propose a name like
Data Acquisition Backbone Framework **DBF** (DataBaseFormat!) or **ABF**
Data Acquisition Backbone System **DBS** (Deutscher Bildungs-Server) or **ABS**
Data Acquisition Backbone Core **DABC** or **ABC**
FAIR Acquisition Backbone **FAB**

3.2 *DABC* architecture

3.3 Implementation phases

One can identify four implementation phases, i.e. time scales:

1. *MBS* front end support
2. Hardware tests and basic functionality of framework
3. Detector test
4. Production system

Phase 0 might be in the beginning necessary for elementary test of the boards and the data links. The software of phase 0 will be "experimental" and not part of the system.

3.4 Requirements

3.4.1 Frontend test bed

Phase 0, 1

Fig. 3.2 shows the hardware components, fig. 3.3 shows the data streams.

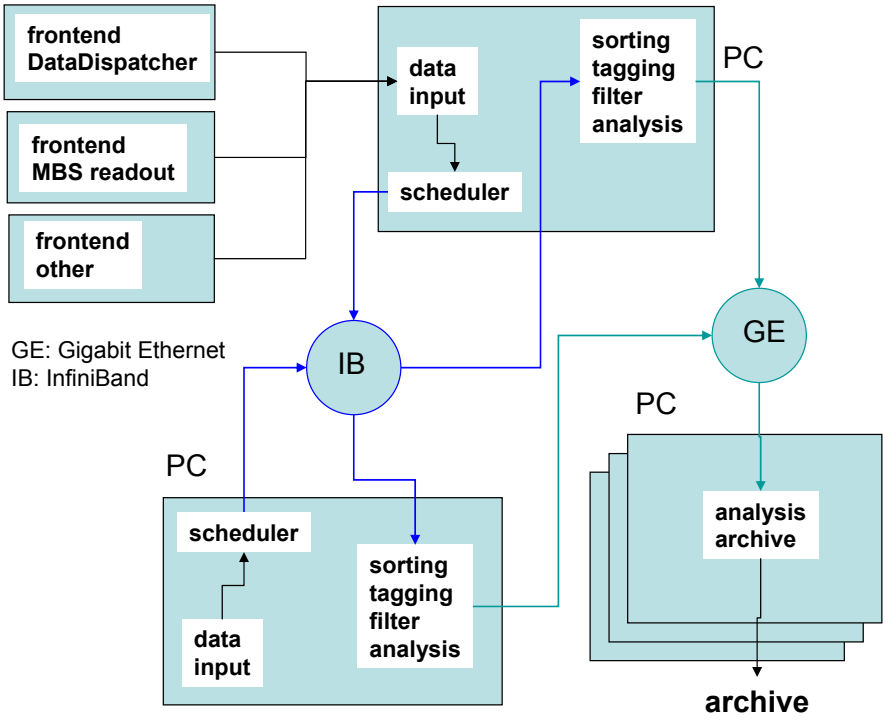

Figure 3.1: *DABC* data flow and components

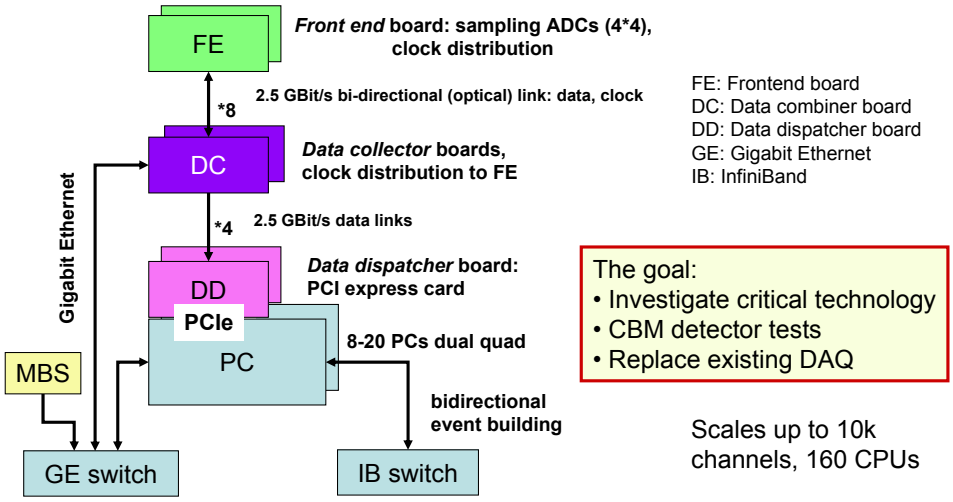

Figure 3.2: *DABC* overall data processing architecture

3.4.2 Detector test bed

Phase 2

3.4.3 Switched event building

Phase 2

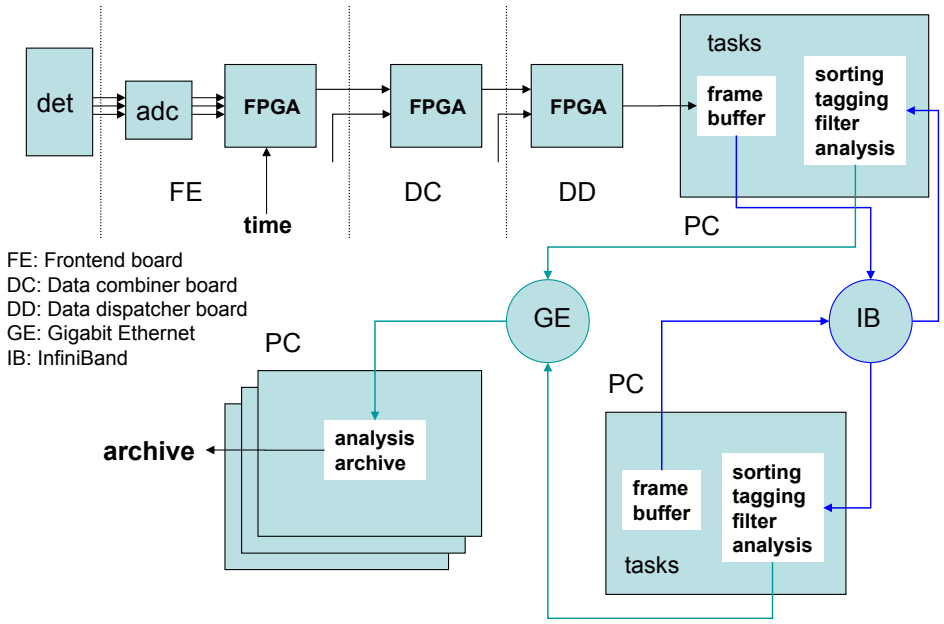

Figure 3.3: Overall data streams architecture

3.4.4 DAQ framework and control systems

Phase 2

3.4.5 *MBS* front-end support

Phase 1

3.4.6 Hybrid setup

Phase 4

3.5 Design

3.5.1 Task layout

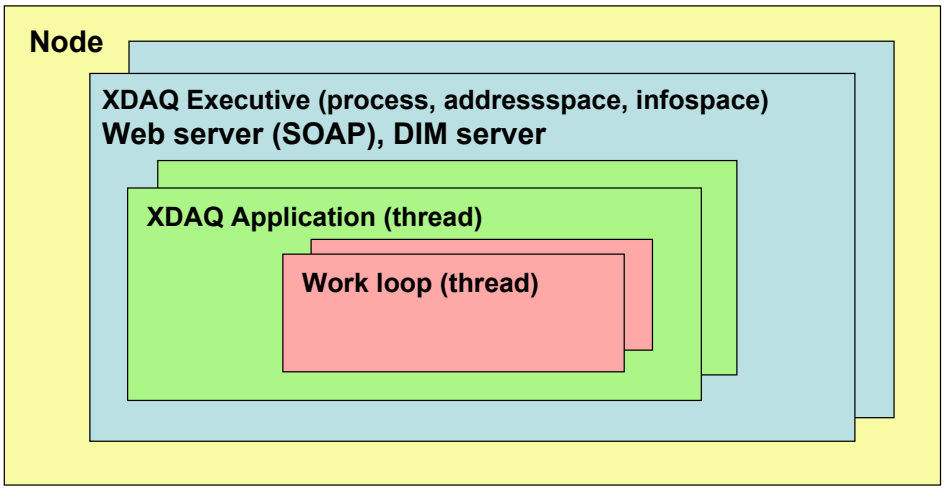

Figure 3.4: χDAQ entities

Figure 3.4 shows the task layout of χDAQ . On each node there is at least one executive which controls the χDAQ applications. The applications run in threads thus being scheduled and sharing memory. Applications also may start and control threads (work loops).
The executive also is the communication stub through the Web interface or the DIM gateway. The executive addressspace is the scope of the infospace. The infospace provides mechanisms to access parameters across applications and subscribe for parameter changes. The DIM gateway subscribes for parameters in the infospace and serves them to DIM clients.
Figure 3.1, page 17 shows the data flow. There are two main χDAQ executives. One runs the applications for the data input from the front-end systems, the event building scheduler and the network senders. The second one runs the event receiver applications for event tagging, event building and event senders.

3.5.2 *MBS* data input

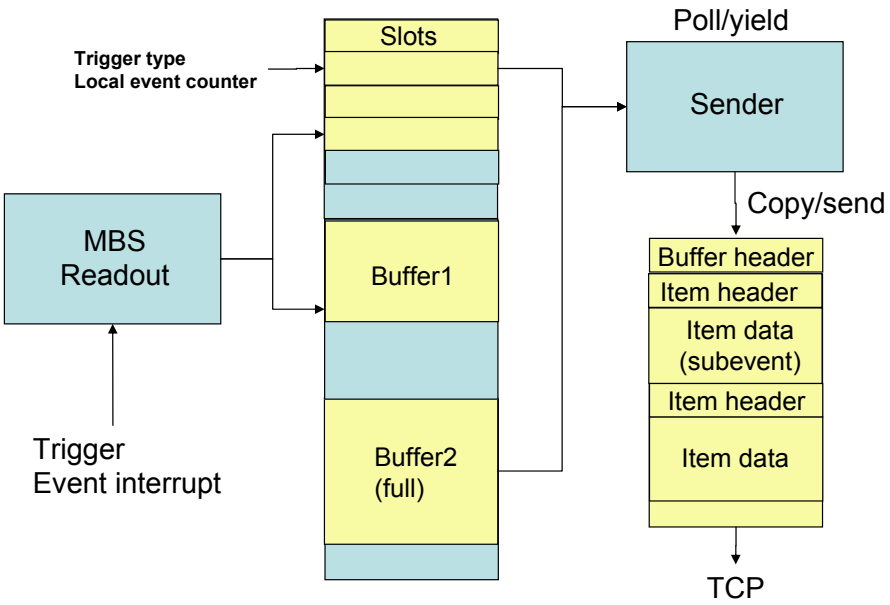

Figure 3.5: *MBS* sub-event sender

An *MBS* data channel delivers sub-event data from one *MBS* readout node. On the *MBS* side the readout task shares the sub-event pipe memory with the sender task (see Figure 3.5). The readout task receives interrupts from the trigger module. Then it checks if there is a free slot in the pipe slot table and enough memory in one of the two buffers. If not it yields to give the sender the chance to empty slots. If there is a free slot and memory, the sub-event data is read from the digitizers into the buffer. The pipe consumer, e.g. an *MBS* collector, polls on ready slots, copies the sub-event data into a buffer, and frees the pipe entry.

3.5.2.1 Copy mode

Instead of the collector one can use a data sender filling a network buffer for sending to the *DABC* data channel (copy mode).
The network buffer begins with a header (endian, size, structure ID, number of items, ...). Each item (sub-event inclusive header) has an item header keeping the trigger type and sub-event counter (from trigger module, cyclic).
The logic of such a mode is very similar to the *MBS* collector/transport (transport mode). Therefore a *DABC* input channel could alternatively connect to an *MBS* transport and read LMD buffers. The disadvantage of this logic is the performance loss due to the data copy. In the first case, the data is more convenient for the receiver, because it's format can be optimized for it. In LMD buffer streams events may span over buffers.

3.5.2.2 Zero copy mode

Another logic would not copy the data but send it directly from the pipe memory (zero copy mode, Figure 3.6). In this case a sub-event directory structure must be sent before. Each entry keeps the trigger type and sub-event counter. In the *DABC* input channel the sub-event data must be combined with the

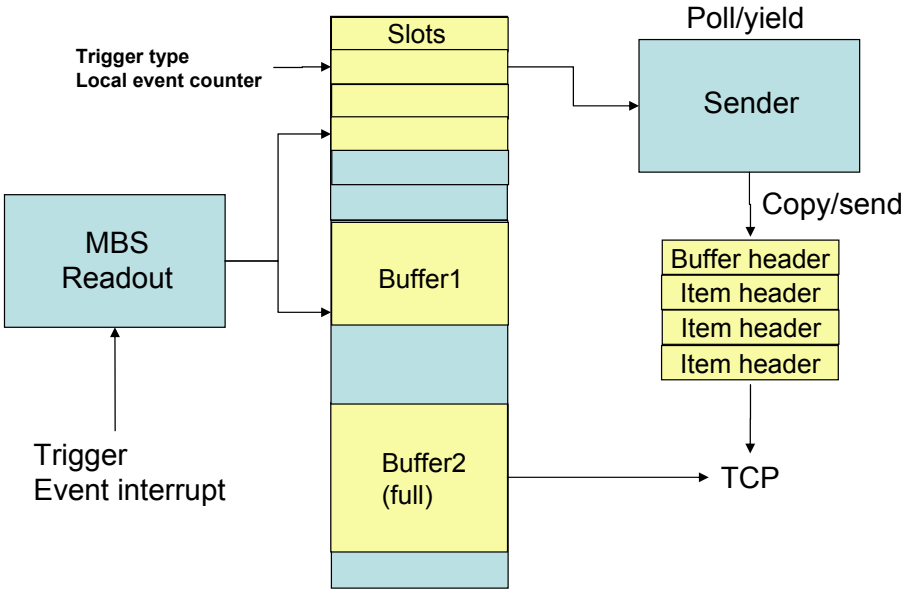

Figure 3.6: *MBS* sub-event sender zero copy mode

directory data, i.e. copied. A similar logic is used by the *MBS* sub-event sender. This sender is already optimized with the readout and could be used as base for a *DABC* sender.
Unfortunately the best performance logic depends on the event rate/fluctuation and the sub-event size/fluctuation. With small sub-events at high rate logic one (copy and one TCP write per many events) would be best, with large sub-events at low rate the last logic (one TCP write per event). At the receiver side the difference is mainly in the data structure. A data copy as required at least with the transport and zero copy mode will be faster than on the front-end CPU. However, several *MBS* input channels may run on one *DABC* node.

3.5.2.3 Communication protocol

1. The *DABC* input channel connects to the *MBS* server, either the transport or the sender (must be started on *MBS*-side) selected by port number.
2. An acknowledge message of 4 longwords is returned (endian=1, size, buffers, streams). For transport that means that *DABC* needs a buffer of size x buffers bytes to store all events of a stream completely (no fragments).
3. Start reading buffers. When the reading is slower than the *MBS* can send, the *MBS* blocks!
   MBS transport sends always all buffers of a stream, except empty buffers and buffers behind the one containing the stop acquisition event. Because *DABC* might need to know when a stream is finished, a change in transport is necessary to mark the last buffer of stream. For this the not used l_free[2] field in the buffer header might be used. Currently it is always 0. A 1 could indicate last buffer. Events must be copied to be contiguous in memory (because of spanning events)
   MBS-sender returns size=maximum, buffers=1,2 (copy, zero copy mode), streams=0. The data buffers it sends have variable length. Therefore *DABC* reads always a minimum buffer (e.g. 1K) to get the actual size, and then the rest. Depending on the mode the buffer read must be processed differently. In copy mode the buffer probably is formatted in a way to be processed directly. In zero copy mode the events must be merged with additional data, i.e. copied.
4. Close connection.

The resulting format of all modes should be identical, i.e. like the copy mode:

Londword	endian=1		
Londword	structure ID		
Londword	size		
Londword	number of items		
Item	Longword	Trigger type	
	Longword	Event counter	
	Subevent	Longword	Size
		Word	Type
		Word	Subtype
		Word	Processor id (from setup)
		Byte	sub-crate
		Byte	Processor type code
		3 Longword	Time stamp (optional)
			Data

Table 3.1: Buffer structure.

3.5.2.4 *MBS* control

The remote *MBS* system can be controlled completely by *DABC*. The *MBS* is started by remote shell commands and controlled by sending commands to the *MBS* prompter. The controlling process can request the *MBS* status information.

4 Hardware

4.1 Hardware Introduction

[Marker:hw.introduction]

4.1.1 Demonstrator

4.1.1.1 Front End Electronics board FEE

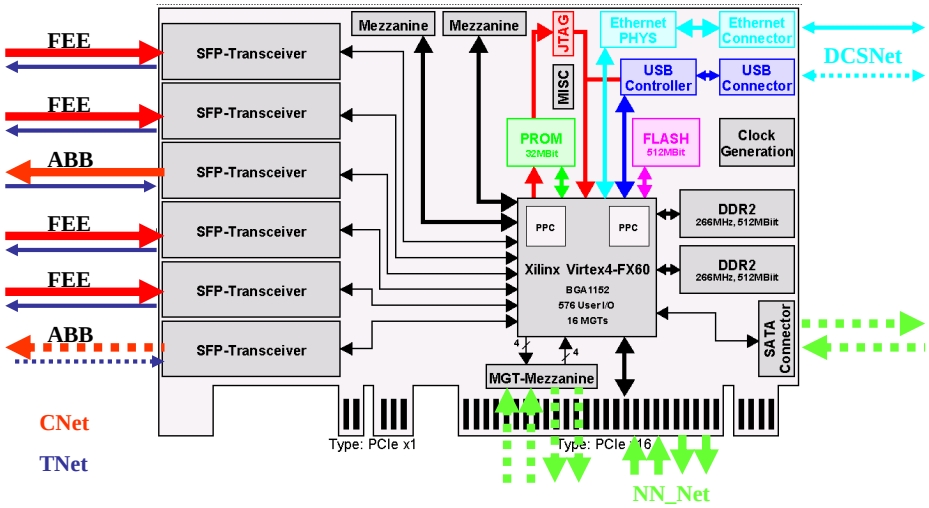

Figure 4.1: Scetch of FPGA board (Brüning)

General purpose chamber-mountable board with ADCs (4*4 channels, 65 MHz sampling, 10-12 bit), FPGA (Virtex-4, PPC, Ethernet MAC, MGT plus SDRAM, CPLD, NVRAM), plugable to preamp/chapers. A version utilizing pipeline TDCs (ns resolution) possible. Two clock domains: receive clock of 312.5 MHz (timing and trigger) and sampling clock with 62.5 MHz (measurement). Four pair of LVDS bi-directional links (625 Mbps).

4.1.1.2 Data Combiner board DCB

Four bi-directional LVDS links to FEB, Ethernet, MGT with SFP optical link to ABB, same FPGA. A test layout of such board is shown in Fig. 4.1. A possible connection to the between FEE and DCB boards is shown if Fig. 4.2

4.1.1.3 Active Buffer board ABB

PCIe board, 4 - 8 MGT/SFP optical links to DCBs, same FPGA.

4.1.1.4 Timing board

Similar to DCB, gets optional trigger inputs, generates clock and distributes through optical splitters to DCBs.

4.1.1.5 Event builder

Standard PC with GE, InifiniBand switch.

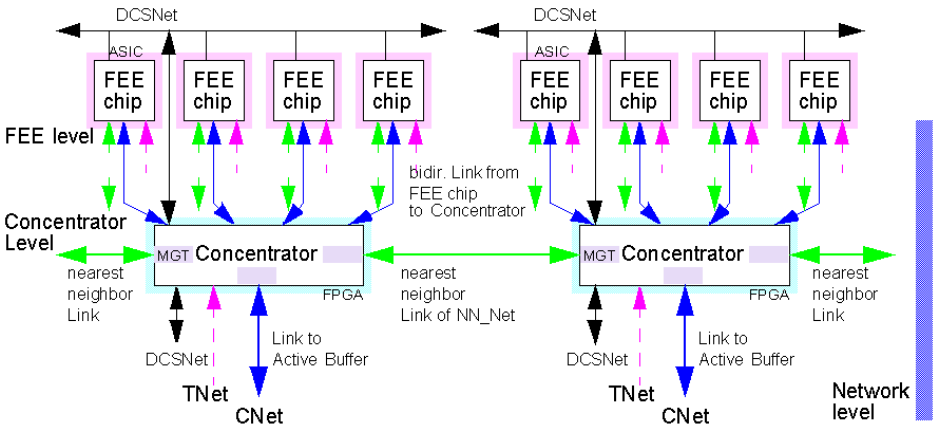

Figure 4.2: FEE and DCB connections

5 Software

5.1 Software Introduction

This part describes the software of the Linux based components. The FPGA software is described elsewhere in detail, but the interfaces are described here, as well as the data formats.

- ◇ Functional components
- ◇ Data streams
- ◇ Communication and control
- ◇ User interfaces

5.2 Data streams overview

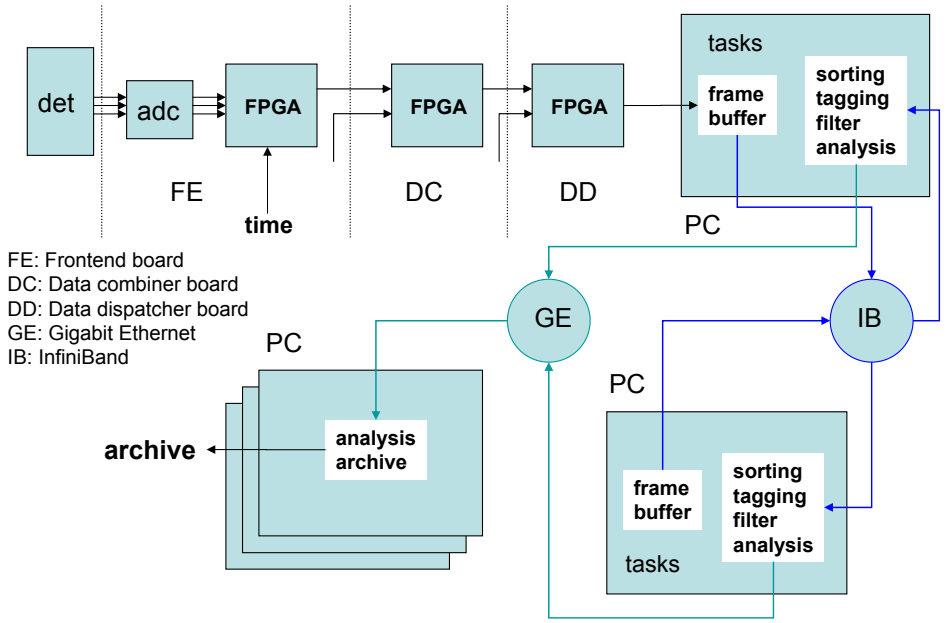

Figure 5.1: Overall data streams architecture

Fig. 5.1 shows the data streams. On the FEE board the data from the ADCs are tagged with time stamps. The data following a time stamp may be a stream of ADC values. Certain time stamps are time epoch markers. Data items sent by the FEE need not to be time ordered, but **all items of one epoch should be in between the epoch markers** defining an *epoch buffer*.
Similarly, the DCB merges all incoming epoch buffers into one output stream where all epoch buffers are subsequent. The ABB collects the epoch data from the inputs (one or two) and stores them in the frame buffer in the PC.
The frame buffer now contains complete streams of epoch buffers containing data items without time order. These epoch streams can be used as switching unit. **It is assumed that all items of an epoch buffer belong to that buffer!** Through the switching all epoch streams of one epoch are combined

on one PC. Now the data items must be reconfigured, i.e. time sorted and eventually reformatted. By multiplicity histograms over time the event time intervals can be calculated. The data items inside this interval build the event which is tagged.

5.3 Data formats

All FEE run asynchronously. Let use term ¨hit¨(whatever it means) as minimum portion of data, which is provided by detector. Hit data size (excluding detector channel id and time stamp) will be about 1-8 bytes depending of detector system. Each hit should be assigned with the time stamp. Most probably, not individual hits but rather group of hits will be assigned with the same time stamp. Required resolution for time stamp depends on detector type. For most detectors 1 ns will be enough, for other 50 ps is required.

5.3.1 Proposal for data format

[Marker: sw.data]
Lets introduce following types for markers:

Value	Name	Marker	description
0000	Epoch	M	Value of time epoch in 10 μs units. 28 bits can provide unique code for each epoch inside $\tilde{2}$500 s time interval.
0001	time shift	T	To achieve 1 ns precision of time shift inside 10 μs epoch, 13-14 bits are required. For the 25 ps precision 19 bits can be used. Probably, one should use rest 9 bits as size indicator how many data assigned to this time marker. This allows fast navigation in packet to find given time value, which is important in time sorting algorithm.
0010	channel identifier	ID	About 10 Mchannels are expected [1] (excluding MAPS). 28-bits code can be divided on two-three groups to have: clear system number like STS, TRD etc, subsystem number STS1, TRD2 etc, channel inside subsystem. Should exists clear algorithm to define size of data for given channel id (something like lookup table).
0011	Event tag	TG	28 bits allows $\tilde{2}$.5 108 codes for events or about 25 sec for repetition of the same code at 107 events/sec rate.
0100	Histogram	H1	28 bits can include initial time shift and size of histogram.
0101	Peak	P	28 bits is used for time shift ($\tilde{1}$3 bits), amplitude (9 bits), width (6 bits)
0110	Schedule	SCH	28 bits used to specify schedule size. Schedule include dependency between time, event tag and event builder, which should receive event data.

Table 5.1: List of data types.

One can imagine a lot of variants to code data to the binary arrays. Probably, one can formulate some rules, which allows more clearly define format. For instance:

- all data flows can be separated by packet of variable size;
- should exists simple algorithm to merge data of several packets into one;
- data of following packet should not depend from previous one;
- all entities of the system should use same format.

Probably, there are other rules, which can be specified. Lets try to define format, which conform to this rule. For instance:

- any data inside package are represented as array of 4 bytes values;
- some of this four-bytes values are defined as markers;
- marker include 4-bit type and 28-bit data parts;
- any packet should be started from such marker value;
- should exists clear algorithm to navigate from marker to marker inside packet;
- data between markers is of any kind binary data, rounded to 4 bytes.

This list (5.1 can be further extended to have unique identifier for any markers in all packets, used in the system. It is supposed, that address information and packet length are provided by transport protocol. Data packet, coming from FEE, can be:

```
M   1        epoch marker
T   17       time shift inside epoch
ID  15       detector channel id, where hit is detected
   100  200  detector specific data
T   68       time shift of next hit
ID  34       detector channel id for next hit
    20
T  134
ID  18       this detector may not require any data
T  135
ID  19
   100  200
```

6 Controls

6.1 Controls Introduction

[Marker:ctrl.introduction]
For the three main fields of detector controls, DAQ controls, and board controls there are currently three products under test:

EPICS Well known control system, widely used in accelarator and classical controls. SNMP interface is available for monitoring clusters and net devices.

LabView Standard slow control at GSI.

SysMES Configuration control system developed at KIP. Has CA interface to work with EPICS and SNMP.

xDAQ DAQ framework of CMS

Because it might be not possible to make a decision for one of these to be used exclusively one should investigate/implement interoperability. Communication standards in question:

- DIM: DIM server in an EPICS IOC as first test bridge
- CA: LabView CA client available
- SOAP: provided by xDAQ, Java interface available.

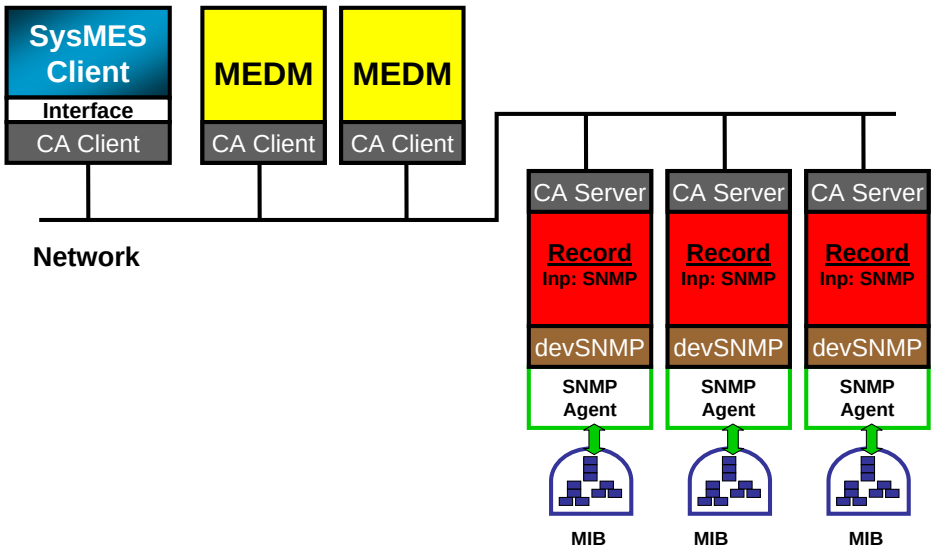

Figure 6.1: SNMP based control architecure

The DAQ controls must provide the following functionality:

- Control tasks on remote machines
- Communicate with all tasks
- Mechanism to store/retrieve the whole setup
- Monitor setup, status, data flow, and performance
- Control data flows
- Visualization and GUI

7 Work packages

Definition of work packages

7.1 Hardware

7.2 Software

7.2.1 Framework

7.2.1.1 GUI

7.3 Controls

8 Simulations

8.1 Simulations

9 Testings

9.1 Overview

This is the introduction of the test chapter. Here we will explain what we test and why.

9.2 Test proposal at FZK

Proposal for InfiniBand testing in FZK

9.2.1 Future DAQ for FAIR

The experiments planned for the upcoming new facility FAIR at GSI require a new generation of data acquisition systems. As an example we consider the Compressed Baryon Matter experiment CBM. The data acquisition for this experiment requires fast building network technologies, capable to transport and switch enormous amount of data in real time. The main task of the event building network (BNet in fig. 9.1 is to combine data of one (or several) events in one computer node, where special filtering algorithms can be performed to accept or reject the event for further physical analysis. These algorithms require almost all data of an event. Therefore a triggerless DAQ is envisioned. All data items are time stamped. The time stamps are then used to define the events and determine the data of an event. Experimental data flow from the many detector subsystems continuously in parallel into the event building net. They should be resorted according their time stamps or event tags. Actually, we need a logically network with many ($\sim$1000) nodes, where each node should be able to exchange data with any other. Beside the main data traffic one expects to have meta data, which should be transported over the same network: flow control and transfer scheduling data. SystemC simulation was done to investigate different possibilities for traffic patterns.

9.2.2 Infiniband cluster in GSI

At the preliminary phase we consider InfiniBand as possible candidate for such a building network. For investigation of InfiniBand and communication protocols a 4-nodes cluster was established at GSI since November 2005. Each node consists of a Double Opteron machine, equipped with Mellanox MHES18-XT (PCIe) host adapters. For connectivity a Mellanox MTS2400 switch was used.

9.2.3 IBGold and uDAPL tests

At the beginning we have installed and use Mellanox IBGold 1.8.0 package. uDAPL was taken as main data transport protocol. The main features of uDAPL are: point-to-point communications, zero-copy data transfer and remote DMA. A C++ library was implemented to wrap the main functionality of uDAPL in more convenient form for usage in test programs.

A mechanism to synchronize the clocks of the nodes was implemented. It uses small round-trip packets between master and all slave nodes. Each round-trip packet takes about 12 μs. The master calculates

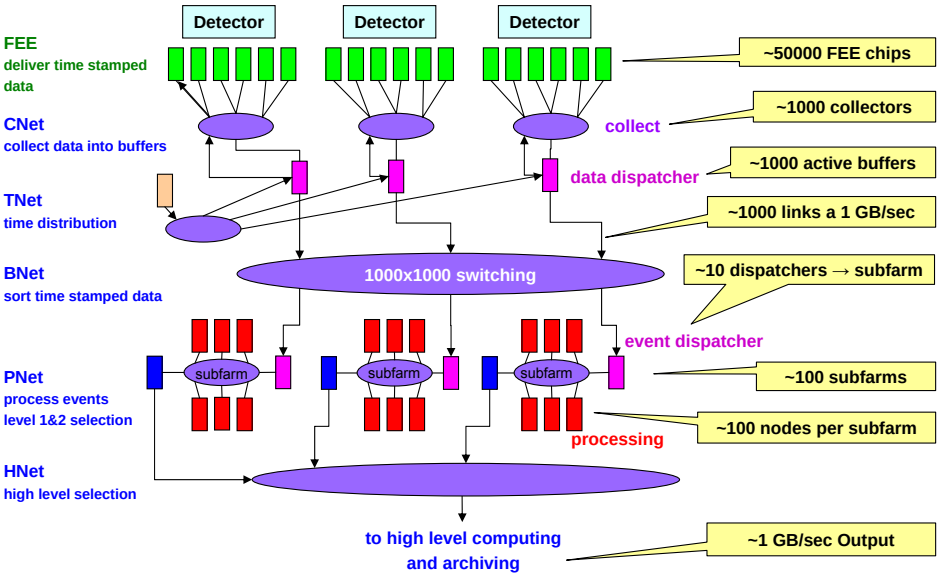

Figure 9.1: CBM overall data processing architecture

the clock difference and submits correction coefficients (time shift and scale) to all nodes at the end. One achieves a clock synchronization between nodes with a precision of a few μs per 100 seconds. For long-time tests the time synchronization routine must be repeated regularly.

A scheduled data transfer mechanism was implemented. Scheduled transfer means that each nodes gets a predefined traffic pattern, where transfer buffer size, target node and exact sending time are specified. In the test applications always full connectivity is established, therefore arbitrary traffic patterns can be tested. Several patterns were investigated: one-to-all, all-to-one, all-to-all (several variations). For us all-to-all performance is of main interest while it corresponds to the expected traffic pattern of the CBM event building net.

We also have investigated non scheduled data transfers. In this case the data transfer is not synchronized between nodes. Each node just sends packets in each direction until the sending queue will be filled completely. The result of both tests are represented in the table 9.1. More detailed results can be found in the following sections.

Buffer	1K	2K	4K	8K	16K	32K	64K	128K	256K
Scheduled	247	437	705	877	910	936	948	953	954
Chaotic	273	524	776	889	923	937	947	953	954

Table 9.1: Receive data rates (in bytes/μs) in IBGold uDAPL all-to-all tests

One can see that above 16K buffers the transfer rate exceeds 900 bytes/μs. One should also take into account, that the total throughput over each InfiniBand host adapter is two times bigger because each node receives and sends data simultaneously. We also observe that the resulting data rate depends from the size of sending and receiving queues (not shown here).

One can see that chaotic transfer (no schedule) is faster for small packets sizes. Actually, with older firmware, we observed a much bigger difference. The advantage of chaotic traffic is that it does not require additional time synchronization between send operations. But it is still not clear if such approach will work on bigger system with many nodes.

9.2.4 OFED and verbs tests

In November 2006 we start investigation of new OFED 1.1 package, provided by OpenFabrics Alliance (openfabrics.org). As Mellanox IBGold package, it includes all required drivers and protocols for working with InfiniBand clsuters. Actually, this package becames an official development line for Mellanox and will replace IBGold in the future.

First we have tested uDAPL from OFED package. Results shown in table 9.2.

Buffer	1K	2K	4K	8K	16K	32K	64K	128K	256K
Scheduled	264	490	700	788	835	866	875	879	880
Chaotic	309	536	688	788	838	868	878	881	882

Table 9.2: Receive data rates (in bytes/μs) in OFED uDAPL all-to-all tests

Results looks similar to IBGold, but there is other upper limit of achievable bandwidth 880 bytes/μs instead of 954. This limitation appears only for all-to-all communication and must be investigated separately. For all-to-one and one-to-all with OFED we achieved data rates up to 985 bytes/μs, which is better than 972 bytes/μs with IBGold.

With OFED we also have tried another library - libibverbs. This protocol allows programs to use InifiniBand "verbs" for direct access to IB hardware from userspace. Main advantage of verbs compared to uDAPL is the support of multicast data transfer, missing in uDAPL API. While uDAPL interface is very similar to verbs, we design common light-weight C++ interface for both uDAPL and libibverbs. This allows us to run same performance tests with "verbs" too. Actually, results of "all-to-all" tests were not much different from uDAPL ones. Only for small packets we achieve better performance - 365 bytes/μs for 1K buffers.

We also extend functionality of our C++ interface to be able work with multi-cast data transfer. Achieved performance of multi-cast transfer was 625 bytes/μs for 2K buffer size. A drawback of multi-cast is the unreliable protocol behind; therefore packet lost should be taken into account. On our cluster we observe maximum 0.002% portion of lost packets for multi-cast-only transfers.

With the multi-cast we were able to test traffic patterns, similar to that was simulated with SystemC. Such pattern mixes normal all-to-all data transfer (90%) with additional flow control traffic (10%). Controlling traffic includes point-to-point slave-controller communications and multi-cast traffic from controller to all slaves. For 8K buffers we achieve 750 bytes/μs of main data traffic and 23 bytes/μs of multi-cast traffic. And we see no multi-cast packets lost.

9.2.5 Planned test in FZK

9.2.5.1 One switch

First, we would like to repeat the same communication tests on a bigger cluster in scope of single InfiniBand switch (switch module). We would like to test, how network performance (data throughput over each connection) depends from the number of active nodes. And also it is interesting to see if chaotic transfer still as efficient as scheduled traffic. 12-24 nodes should be enough for such kinds of test.

9.2.5.2 Switch fabric

Second, we would like to test all-to-all communication on a switch fabric, where more than one switch (switch module) is involved. In that case it is interesting if correct traffic scheduling gives an advantage

or if one can keep a simple round-robin scheme or even non-scheduled transfer at all.

9.2.5.3 Big cluster

Third, our special traffic pattern with small portion of multi-cast traffic would be interesting to test on a relatively big system. Do we get same performance as for small system, how much multi-cast packets will be lost, can we implement simple retry algorithm for them?

9.2.6 System requirements

For usage of uDAPL (in IBGold or OFED) following components are required:

- uDAPL library - libdat.so
- uDAPL includes - "dat/udat.h"
- configured IPoIB - required by uDAPL to establish connections
- user account and ssh (or rsh) to run application

For usage of libibverbs and multi-cast in OFED:

- verbs library - libibverbs.so
- verbs includes - "infiniband/verbs.h"
- Open Subnet Manager (OpenSM) libraries - libopensm.so libosmcomp.so losmvendor.so
- opensm includes - "vendor/osm_vendor_api.h" "vendor/osm_vendor_sa_api.h"
- user account and ssh (or rsh) to run application

9.3 Testing the UDAPL library

Here we describe what we did with the uDAPL on the IB cluster.

9.3.1 Overview

uDAPL is a user-level direct access API developed by DAT collaborative `http://www.datcollaborative.org`

The main objectivity of uDAPL is to provide a transport and platform independent API for data communication between nodes. The main features of uDAPL are:

- **point-to-point** connection, advanced connection management;
- management of **memory** buffers;
- **zero-copy** data transfer;
- **message** and **RDMA** data transfer.

9.3.2 Installation on IB test cluster

The mellanox IB Gold distribution v 1.8.0 for Linux was installed on the Infiniband-cluster (see `https://docs.mellanox.com/dm/ibgold/ReadMe.html` for more details). Source package and documentation can be found in `/usr/oub/ibgd`.

The package is installed localy for each node in `/usr/local/ibgd`. The complete set of components was installed. To compile Mellanox IB Gold, package `termcap-2.0.8-879.i586.rpm` (from SuSE distribution) was installed on each machine.

After installation several configuration files where modified:

Table 9.3: List of modified configuration files

`/etc/infiniband/ifcfg-ib0`	here correct IP address for Infiniband should be set
`/etc/infiniband/openib.conf`	set modules, which should be loaded, enable uDAPL (default - no)
`/etc/opensm.conf`	on master node ONBOOT=yes should be set

IP over IB (IPoIB) was configured in following way:

Table 9.4: List of IP adresses in InfiniBand network

Host	IP adress
master	11.0.0.1
node01	11.0.0.2
node02	11.0.0.3
node03	11.0.0.4

9.3.3 Running of standard InfiniBand transport tests

Included in the Mellanox IB Gold distribution are test suites for MPI and uDAPL stack. With them one can test bandwidth and latency for both protocols.

9.3.3.1 MPI tests

They are located in `/usr/local/ibgd/mpi/osu/gcc/tests` (which can be accessed as `$MPITESTS`).

1. MPI bandwidth test (repeat=1000, packetsize=1000000):
`mpirun_rsh -np 2 node01 node02 'echo $MPITESTS'/osu-tests/bw 1000 1000000`
Result is:

```
1000000 952.753887 Mb/s
```

2. MPI latency test:
`mpirun_rsh -np 2 node01 node02 'echo $MPITESTS'/osu-tests/lt 1000 16`
Result is (μs):

```
16      4.025500
```

3. Presta bandwidth tests between two nodes:
`mpirun_rsh -np 2 node01 node02 'echo $MPITESTS'/presta1.2/com -o 100`
Result is:

```
Max Unidirectional Bandwith :  953.18 Mb/s for message size of 8.4 Mbytes
Max  Bidirectional Bandwith : 1479.47 Mb/s for message size of 0.5 Mbytes
```

4. Presta latency test:
`mpirun_rsh -np 2 node01 node02 'echo $MPITESTS'/presta1.2/laten -o 1000`
Result is (μs):

```
Maximum latency = 4.947
```

5. Presta test for MPI_Allreduce operation:
`mpirun_rsh -np 4 master n1 n2 n3 'echo $MPITESTS'/presta1.2/allred 10 10 100`
Result is (times in μ):

```
MPI Allreduce test
  for 4 MPI processes, message size 32 bytes,
  10 operations per sample, 100 samples

  Wtick resolution           :      10.000
  Time between Allreduce     :       7.000

  Mean Compute Loop Time     :      65.270
  Ticks per Compute Loop     :           7
  Mean Op Loop Time          :     179.200
  Ticks per Op Loop          :          18

  Op mean                    :      11.393
```

6. Presta MPI Global-Op benchmark:
`mpirun_rsh -np 4 master n1 n2 n3 'echo $MPITESTS'/presta1.2/globalop`
Result is (μ):

```
Average elapsed run time was    4.21
```

7. Pallas tests:
`mpirun_rsh -np 2 node01 node02 'echo $MPITESTS'/PMB2.2.1/PMB-MPI1`
Results:
Are not listed here, can be found on daq4fair wiki.

9.3.3.2 uDAPL tests

The test suite was downloaded from DAT collaborative web site. Seems to be, the same is included in the Mellanox as well.
Currently the example is compiled in `/u/linev` directory, but also can be accessed by other users.
To run the example copy executable from
`/u/linev/dapl/test/dapltest/udapl/Target/dapltest`
to home directory. On node01 run example as server:

```
./dapltest -T S -D ib0
```

On node02 (or any other) run example as client:

```
./dapltest -T T -s 11.0.0.2 -D ib0 -i 1000 -t 2 -w 1
client SR 1048576 server SR 1048576 client RR 100000 client RW 100000
```

Crutial here, that correct IP address of server (for node01 it is 11.0.0.2) should be set.

More details about arguments can be found in Readme file.

Results is:

```
Server Name: 11.0.0.2
Server Net Address: 11.0.0.2
DT_cs_Client: Starting Test ...
----- Stats ---- : 2 threads, 1 EPs
Total WQE        :    1694.91 WQE/Sec
Total Time       :       4.71 sec
Total Send       :    2097.15 MB -     444.31 MB/Sec
Total Recv       :    2097.15 MB -     444.31 MB/Sec
Total RDMA Read  :     200.00 MB -      42.37 MB/Sec
Total RDMA Write :     200.00 MB -      42.37 MB/Sec
DT_cs_Client: ====== End of Work -- Client Exiting
```

9.3.4 C++ wrapper for uDAPL fuctionality

uDAPL is a C-based library and has a number of different structures, which should be initialized before data transport can be started. To simplify futher development of the uDAPL communication test program, C++ wrapper classes for most important uDAPL functions and structures are implemented. These are:

- **TEndPoint** - contains handle and attributes of single point-to-point conection;
- **TMemorySpace** - memory region plus neccessary structures for uDAPL zero-copy data transfer;
- **TMemoryPool** - pool of TMemorySpace objects;
- **TBasic** - combines necessary structures and functions for uDAPL communications.

Central class is TBasic. It provides following functionality:

- initialisation / finalisation of different uDAPL structures, required on each node;
- high-level method for establishing connection with other nodes;
- allocation of memory / memory pools;
- sending / receivng data over uDAPL connection;
- handling RDMA read / write operations.

9.3.5 uDAPL test application

Using the C++ wrapper classes for uDAPL a test application was implemented. The same executable should run on each node. One executable became master role, all other run as slaves. The master drives the complete test, sending commands to each slave one by one. Slaves normally should wait for next command and, when it arrives, start execution. But when real test is executed (for instance, all-to-all data transfer test), absolutely the same code will run on master and slaves. When a command (running test) is completed, the slave again switched to waiting state while master decides which command should be executed next. The command list is defined in the test application and includes:

- time synchronisation;
- sleep;
- exit;
- all-to-all communication test with message data transfers;
- all-to-all communication test with RDMA data transfer.
- chaotic (non-scheduled) data transfer.

The test executable should be started practically simultanious, while during initialisation phase connection with all other nodes should be established. After that slaves just going to command waiting loop, while on master the programmed test sequence is executed. Typically such sequence includes different combinations of packet size, schedule pattern, queues size and so on. The main task of such tests was to determine the maximum data transfer rate achievable with an InfiniBand network with different setups.

9.3.6 Time synchronisation

Time synchronisation between nodes is required to implement a scheduled data transfer. InfiniBand provides 10 Gbit/sec data rate. To synchronise the transfer of two packets of 1 KB size one needs a precision in packet send time of at least 1 μs. It is assumed that at least one or several packets can be buffered in network switches / host adapters. Therefore a synchronisation better than one packet size is not required. For time measurements in a μs range one can use the PC clock counter. In the test application function `ntp_gettime()` is used.

But to be able produce synchronous operations on different nodes, the clocks must be synchronised. For that a small round-trip packet is used. The master sends a small packet with a time stamp inside to one of the slaves. The slave adds its own time stamp value and sends packet back. When the master receives such packet, it can determine how long the round-trip took place and what was a middle time on master and slave sides. As a result, it can calculate the time difference between the time on master and slave nodes. Typical round-trip times are about 29 ± 0.5 μs. Such round-trip packets are sent up to 100 times to calculate the average time difference and to achieve a precision of about 0.2 μs. At the end the master sends a packet to the slave with a time correction value, which should be used to calculate correct time stamps on the slave side.

Time measurements with `ntp_gettime()` function use the CPU clock. On different nodes one can expect a deviation between the CPU clock frequency. On our test cluster 10 s after synchronisation such deviation leads to 200 μs difference between nodes and this difference poportianally grows with time.

It means, that the observed difference in CPU clock is 10^{-5}. This is a very significant value relative to required precision of 1 μs during at least 100 s or 10^{-8}. To compensate this effect, a time scale factor must be applied. Therefore the master repeats time synchronisation after 10 seconds and defines a time scale factor for each slave. With the last round-trip packet this factor is sent to slave.

After the time synchronisation procedure the time stamp on each node is calculated by the following equation:

$$T_{synch} = (T_{clockl} - T_0) * Coef + Shift, \tag{9.1}$$

T_{synch} : synchronised time
T_{clockl} : local CPU clock
$Coef$: time coefficient, applied at T_0
$Shift$: time shift.

The procedure allows to synchronise the time between nodes with a precision of 0.5 μs. After 100 sec the time difference is typically not more than 2 μs. When a test runs longer than 100 seconds, the time synchronisation can be repeat to adjust clocks again.

9.3.7 Scheduled data transfer

Main motivation for tests with the InfiniBand cluster was to investigate the capability of InfiniBand network to transfer data with traffic patterns corresponding to future data acquisition systems. In future DAQ one expects a traffic pattern, where each node should regularly transfer approximately the same amount of data to all other nodes (one after another). Previous simulations of such a network with SystemC [12] simulation tools showed, that to achieve a maximum data rate over a generic network, one should use a synchronous (scheduled) transfer, adjust packets sizes and avoid junction of packets during transfer.

For testing of such a scheduled transfer over an InifiniBand network an all-to-all communication test was implemented. It contains several stages:

1. calculation of the schedule for each node on master,
2. distribution of the schedule between all slaves,
3. allocation of required memory pools and
4. execution of schedule on each node.

The schedule for each node includes sequences of send and recieve opearations, where the time of operation, packet size and destination (source) node are specified. Several traffic patterns can be coded by such schedule mechanismus:

- All-to-one communication, when three nodes send data sequentially to single node;
- One-to-all communication, when single node send data sequentially to other three nodes;
- All-to-all round-robin, when each node sequentially sends data to all other;
- All-to-all fixed-target, when each node sends data only to one other node.

Two different transfer modes were tested:
message transfer and
RDMA (remote direct memory access) transfer.

In the transfer of a data message from one node to another both nodes are involved. First, the receiver should call `TBasic ReceiveData()` method, specifying the buffer where the received data should be stored. Only then the sender can call `TBasic SendData()`. Many `ReceiveData() / SendData()` calls can be queued in uDAPL queues, but in any case the number of packets in the

receiving queue always must be bigger than number of packets in the sending queue.

The RDMA transfer requires a preparation phase, when memory for RDMA operation allocated and all involved nodes should obtain descriptors (handle) of this memory. These memory handles are than used in RDMA operations. RDMA operations destinguish two sides:
RDMA-provider and RDMA-consumer.
RDMA-provider (server in other sense) must allocate memory and provide to RDMA-consumer (client) uDAPL-handles for these buffers. After that the RDMA-consumer can invoke `Post_RDMA_Read()` / `Post_RDMA_Write()` methods to access memory on provider side. At the time of an RDMA transfer the provider has nothing to do.

In all tests RDMA-write operations were used, meaning that the sender was RDMA-consumer and the receiver was RDMA-provider. In case of biderectional transfer both roles were applied. The advantage of using RDMA is that during RDMA-write operations no receiving queue is required. The disadvantage of RDMA is that on receiver side one obtains no any singal from uDAPL when data arrived. To overcome this problem, the receiver must time-to-time check (poll) if data really arrived, which makes it difficult to use RDMA-only transfer. Actually, the recommended way of using RDMA in this situation that after a big RDMA-transfer a small message must be sent to inform the receiver, that data are there.

Results of the message-based test are shown in table 9.3.7 and drawn on the figure 9.2.

Table 9.5: Data rates Mb/s for message-based transfers

Buffer	Round-robin	Fixed target	One-to-all	All-to-one
1K	93	165	360	374
2K	202	292	773	615
4K	416	539	953	968
8K	731	713	955	971
16K	853	828	955	972
32K	922	891	956	972
64K	947	923	956	972
128K	953	938	956	972

The results obtained with RDMA transfer are shown in table 9.3.7 and figure 9.3.

Table 9.6: Data rates Mb/s for RDMA-based transfers

Buffer	Round-robin	Fixed target	One-to-all	All-to-one
1K	110	303	194	320
2K	214	563	389	596
4K	424	766	950	966
8K	788	832	953	969
16K	888	888	954	970
32K	941	928	955	971
64K	951	945	956	972
128K	954	945	956	972

As can be seen the only difference between message-based and RDMA transfer shows up in the range of small packet sizes, where RDMA transfer is two times faster.

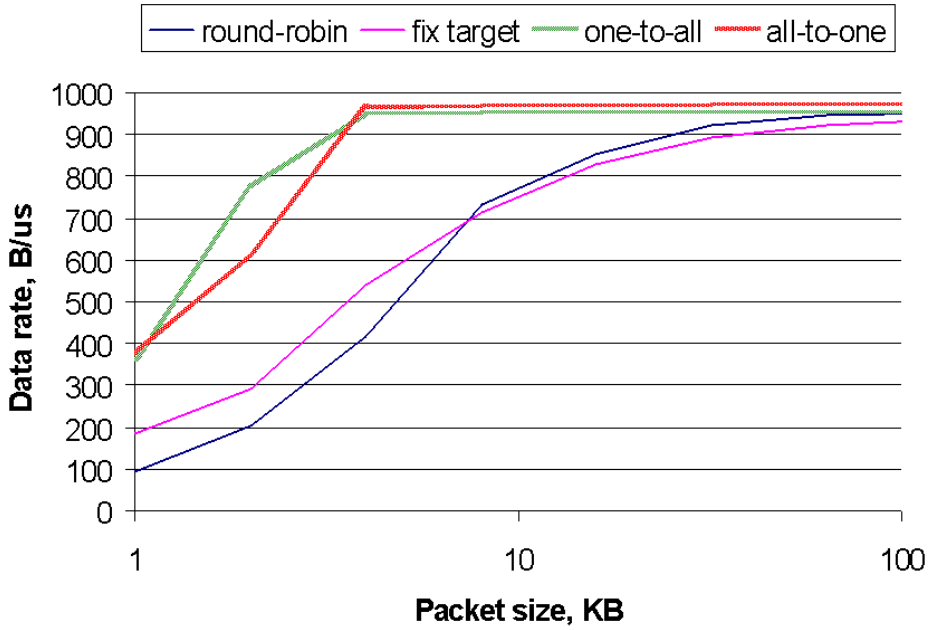

Figure 9.2: Data transfer rates for message-based scheduled transfer

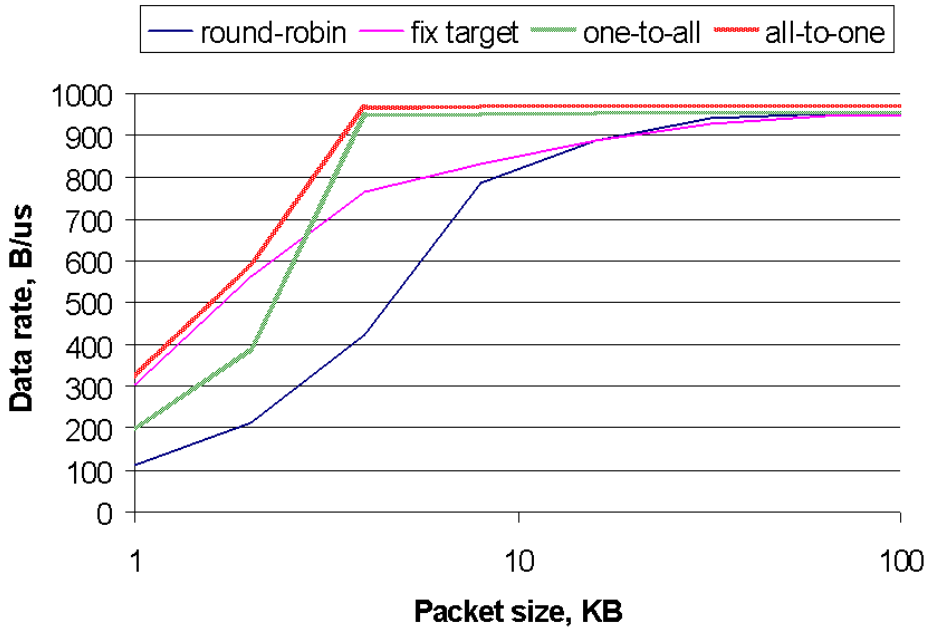

Figure 9.3: Data transfer rates for RDMA-based scheduled transfer

9.3.8 "Chaotic" data transfer

This is opposite to the scheduled data transfer approach. In this mode no synchronisation between nodes is applyed - each node tries to send as many packet as possible in all directions. In practice it means, that the output queue for each point-to-point connection is kept always filled. Still, one should take care, that the receiving queue size is not less than the sending one.

It turns out that the transfer rates depend from queue size allocated for receiving and sending data. The

sending queue size was fixed to 10 entries. For the receiving queue three different values were tested:
11 entries (marked as “queue11”),
20 entries (marked as “queue20”) and
100 entries (marked as “queue100”).
The results of tests are shown in table 9.3.8 and figure 9.4.

Table 9.7: Data rates for chaotic transfers (to be changed)

Buffer	“queue11”	“queue20”	“queue100”
1K	93	142	266
2K	131	168	551
4K	180	490	755
8K	279	909	839
16K	930	948	907
32K	955	957	938
64K	960	959	967
128K	962	962	959

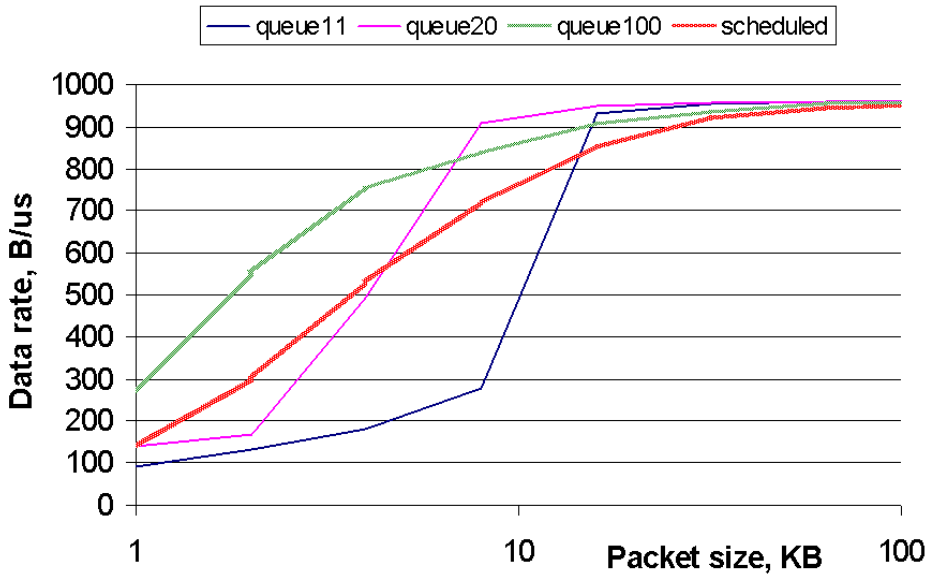

Figure 9.4: Data transfer rates for chaotic transfer compared with scheduled approach

One can see, that for small packet sizes large receiving queues provide better performance. But for buffer sizes above 8 KBytes medium and even small queue sizes gave better performance.

On the figure 9.4 one can also see a comparison of “chaotic” transfers with scheduled ones. From first glance one expects that the data rate achievable by “chaotic” transfers must be less compare to scheduled traffic. But tests with our small 4-nodes traffic show, that the “chaotic” approach provides much better performance for small packet size.

To better understand such behaviour the transfer rates as function of time were investigated. One can see on figure 9.5 that data rates, measured for 1Kb buffer transfers for each connection, change with time. The variation for a single connection can be estimated as about 50%. There are also periods, when total throughput drops down. At the same time measurements, done for 16Kb buffers show practically no any significant variation neither for single connections nor for the sum tranfer rate.

Several explanations could be found for this effect. Physiscally, one has a single InfiniBand cable, connecting host PC and switch, where three virtual connections are established. On a host PC each connection is treated similary. But it seems that on the switch side this is not the case. The switch buffers as many data as it can and push them further with its own algorithms. As a result one sees variations of the transfer rate over single connections and even of overall performance. It seems that scheduled data transfers can only work on the lower limit, which leads to significant loss of practically achievable performance for small packet size.

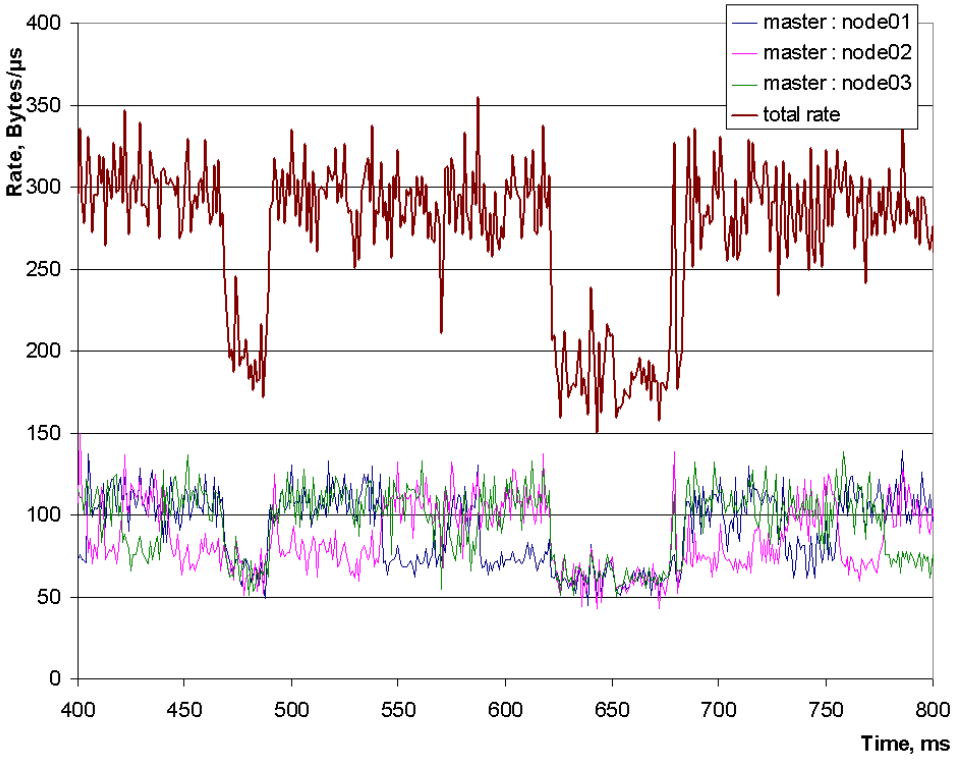

Figure 9.5: Dependency of data transfer rates over P2P connection over time. Buffer 1Kb

The advantage of using “chaotic” transfers is that one does not need an exact timing for send operations and can use uDAPL wait functions instead polling over incoming events. This reduces the CPU usage, especially for big packet sizes, when the number of I/O operations is moderate.

“Chaotic” schedule also was tested with non-fixed packet size. Packet sizes were varing around an average value by $\pm 50\%$. No significant difference with results for fixed-packet sizes were observed.

9.3.9 Concequence of firmware update

After firmware update on host adapter and InfiniBand switch test results were improved. First of all, scheduled transfer with small packets size gets practically the same perfromance as in chaotic transfer. Second, irregularity in the data rates during chaotic transfer practially dissapear.

On figure 9.6 one can see, that transfer rate variation remains arround same average value and does not droped any longer.

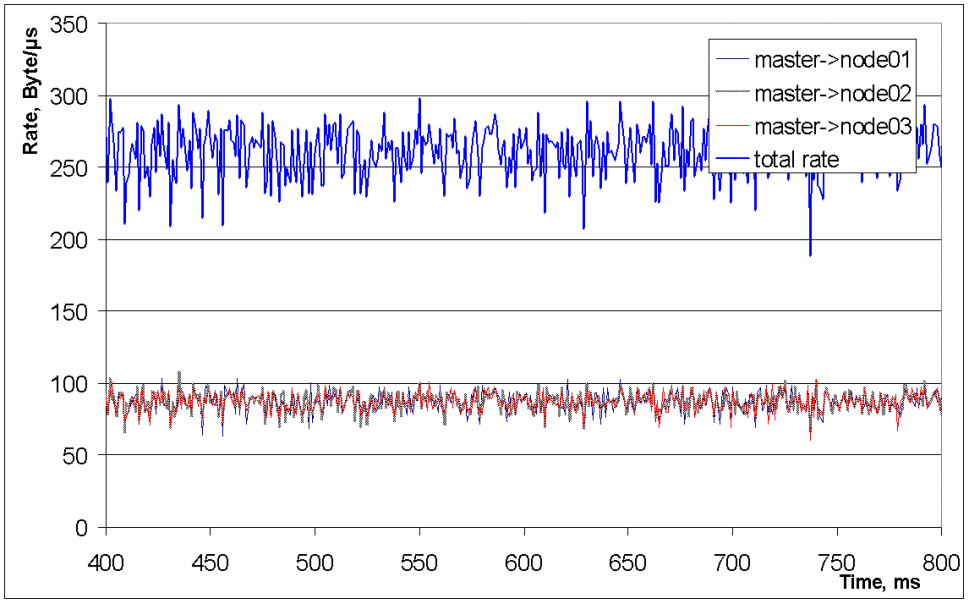

Figure 9.6: Dependency of data transfer rates over P2P connection over time after firmware upgrade. Buffer 1Kb

Glossary

FEE Frontend electronics board

References

[1] CBM collaboration, “CBM Experiment: Technical Status Report”, Januar 2005

[2] CMS collaboration, http://cmsinfo.cern.ch/outreach/, “CMS Outreach”, CERN 2006

[3] The Experimental Physics and Industrial Control System website, http://www.aps.anl.gov/epics/index.php, Argonne National Laboratory 2006

[4] C. Gaspar et al., "DIM - Distributed Information Management System" , http://dim.web.cern.ch/dim/, CERN May 2006

[5] Y. Liu and P. Sinha, “A Survey Of Generic Architectures For Dependable Systems”, IEEE Canadian Review, Spring 2003

[6] The National Instruments Labview web site, http://www.ni.com/labview/, National Instruments Corporation 2006

[7] L. Orsini and J. Gutleber, “The XDAQ Wiki Main Page” http://xdaqwiki.cern.ch/index.php, CERN 2006

[8] L. Orsini and J. Gutleber, “I2O Messaging” http://xdaqwiki.cern.ch/index.php/I2O_Messaging , CERN 2006

[9] L. Orsini and J. Gutleber, “XDAQ Monitor application” http://xdaqwiki.cern.ch/index.php/Monitor_CGI_interface , CERN 2005

[10] L. Orsini and J. Gutleber, http://xdaqwiki.cern.ch/index.php/Configuration_schema “XDAQ XML configuration schema”, CERN 2006

[11] D. Stenberg et al., The curl and libcurl web site, http://curl.haxx.se/, HAXX HB 2006

[12] SystemC website, http://www.systemc.org/

[13] The W3C Consortium, “SOAP Version 1.2 Part 1: Messaging Framework”, http://www.w3.org/TR/soap12-part1, W3C Recommendation, 24 June 2003

[14] K. Whisnant, R.K. Iyer, Z. Kalbarczyk, and P. Jones, “The Effects of an ARMOR-based SIFT Environment on the Performance and Dependability of User Applications”, University of Illinois, 2006

[15] The Wikipedia, “Finite State Machine”, http://en.wikipedia.org/wiki/State_machine, Wikipedia 2006

Index