COMPASS iFDAQ Software

J. Novy

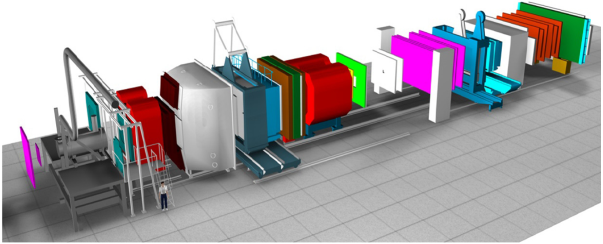
Faculty of Nuclear Sciences and Physical Engineering
Czech Technical University in Prague, Czech Republic
&
European Organization for Nuclear Research – CERN, Switzerland

Outline

- iFDAQ architecture
- I-P(Inter-Process) Communication
- iFDAQ Debugging
- iFDAQ Stability
- iFDAQ Future

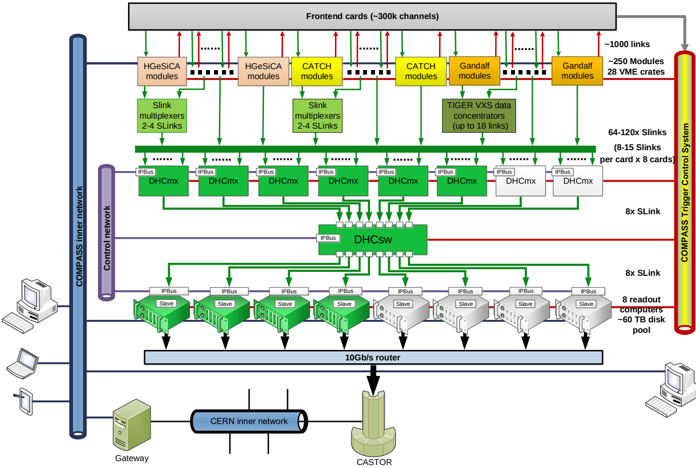
COMPASS experiment

- Fixed target experiment at SPS accelerator at CERN
- Study of hadron structure and hadron spectroscopy with high intensity muon and hadron beams
- Data-taking started in 2002
- Trigger rate up to 40 kHz, average event size up to 50 kB
- In spill data rate 1.5 GB/s and sustained data rate 500 MB/s



Hardware Structure

- Hardware based E.B.
- Data concentrated by 6 (up to 8) DAQ modules with multiplexer firmware
- Distribution to 4 (up to 8) readout computers by DAQ module switch firmware
- Full events received by servers
- Consistency check at many layers
- Events checked and transferred to DATE data format

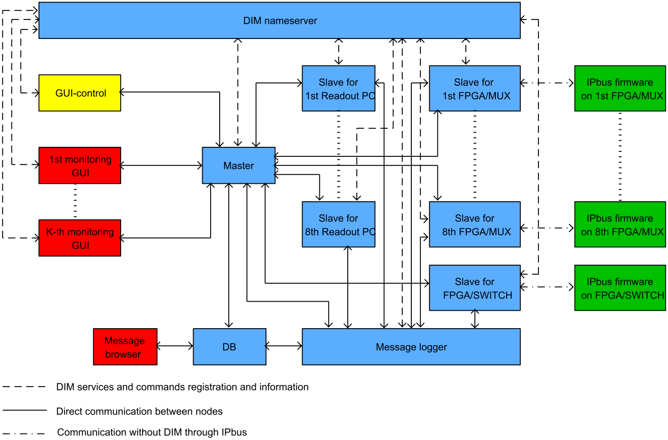


Used Software Technologies

- C++, Python
- Qt framework
- DIM (Distributed Information Management System)
- DIALOG library
- IPbus suite for communication with FPGA cards
- MySQL
- PHP, JavaScript
- Zabbix

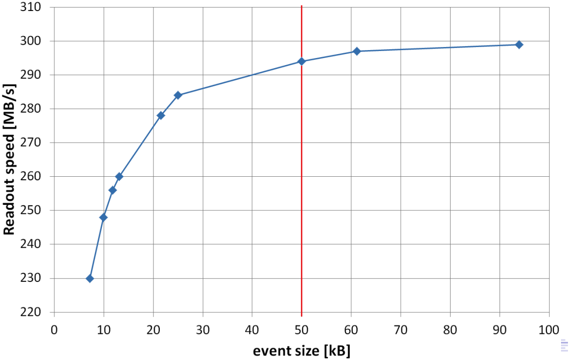
Software Structure

- Runcontrol GUI is a graphical user interface
- Master is a main control process
- Slave-readout readouts and verifies the data
- Slave-control monitors and controls the FPGA cards
- MessageLogger stores informative and error messages into the database
- MessageBrowser provides an intuitive access to messages stored in the database



Maximum Data Readout Performance

- Currently limited by SWITCH firmware to 100 MB/s per RE if 4 RE connected
- If 2 RE connected to not-shared ports, readout speed of up to 150 MB/s per RE



DIM I – Distributed Information Management System

- Developed at CERN in 1993, still with support
- Design requirements
  - Efficient communication mechanism – asynchronous behavior, sending and receiving asap
  - Uniformity – all processes use the same communication mechanism
  - Transparency – any running process should be able to communicate with any other process
  - Reliability and robustness – system recovery in a self-recoverable manner from error situations
- It uses UDP protocol for message transmission

DIM II – Usage in iFDAQ

- Fully incorporated to all processes for the runs 2014 and 2015
- DIM problems
  - High probability of the message truncation or complete loss of the message
  - As a consequence of that, processes crashed without any obvious reason (especially Master process)
- DIM library replaced by DIALOG library
- DIM library can not be completely avoided (still partially present)
  - VME computers have only 64 MB memory $\rightarrow$ we can not install Qt framework there
  - DIALOG library is implemented in Qt framework

DIALOG Library I

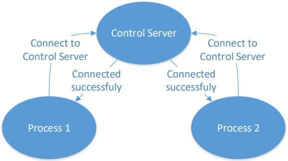
- Replacement of DIM Library
- It is implemented in Qt framework
- Dialog means conversation, talk or speech (**D** – distributed, **I** – inter-process, **A** – asynchronous, **L** – library, **O** – open, **G** – general)
- Design requirements similar to DIM Library
- Communication based on the publish/subscribe method
- It uses TCP/IP protocol for message transmission

DIALOG Library II

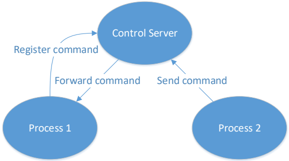
- Services
  - A service is a set of data of any type and size with an unique name
  - Server/Client mechanism – server publishes data to several clients
- Commands
  - Process registers command with a non-unique name it is willing to accept
  - One process can control another one via command
- Implementation
  - The Control Server – keeping an up-to-date list of all the processes, services and commands
  - Providers – a process providing services and commands it is willing to accept
  - Subscribers – a process specifying the service name it is interested in and requesting for it
  - Any process can be a provider and a subscriber at the same time

DIALOG Library III - Scenarios

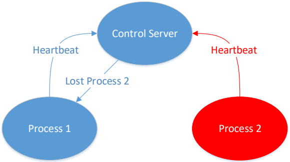
Connection to the Control Server



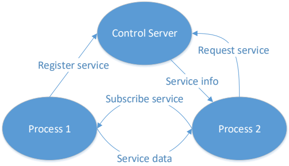
Commands



Heartbeats

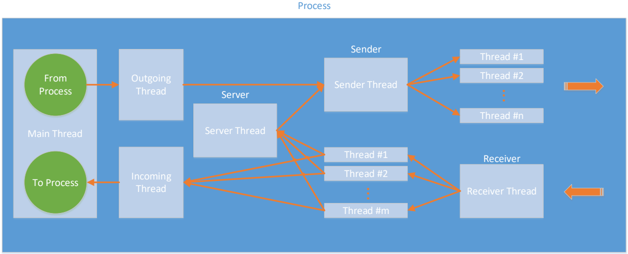


Services



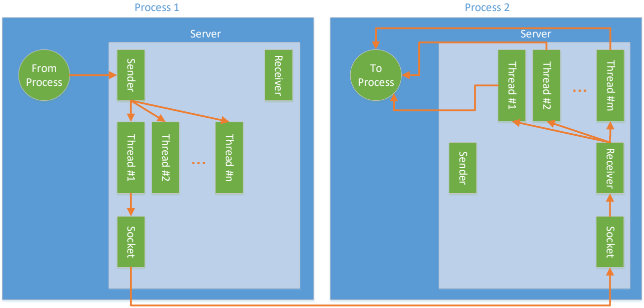
DIALOG Library IV - Process Threads

- Message types are distinguished by message header
- The Sender dispatches messages among $n \in \mathbb{N}$ threads
- $n \in \mathbb{N}$ threads are establishing connections to other processes, writing data to sockets and keeping sockets open until timeout
- Open socket, pointers to messages, sending as soon as possible $\rightarrow$ speed up the performance and reduce the latency significantly
- The Receiver dispatches a new socket descriptor to one of $m \in \mathbb{N}$ threads
- $m \in \mathbb{N}$ threads are responsible for reading data out from sockets
- The sockets are kept open until they are closed by sender process

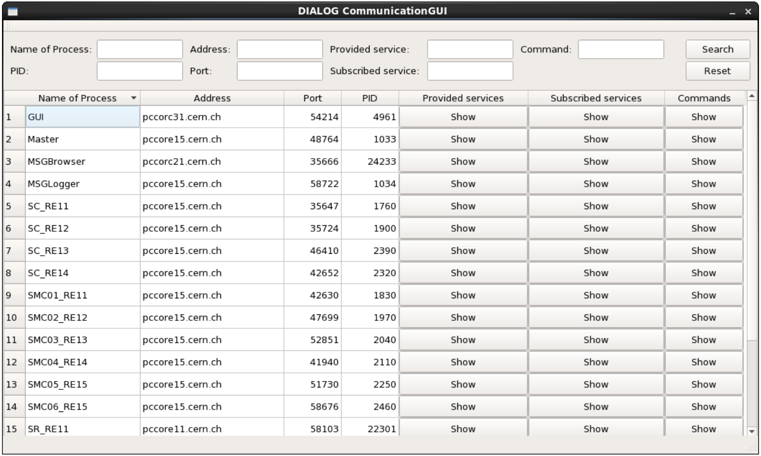


DIALOG Library V - from Process 1 to Process 2

- If the connection is not yet established, the object socket is created and opened in Process 1
- The Receiver receives the socket descriptor trying to connect to Process 2
- Socket objects exist on both sides till timeout, process crash or process termination
- The open socket is used only for one direction connection



DIALOG Online Monitoring

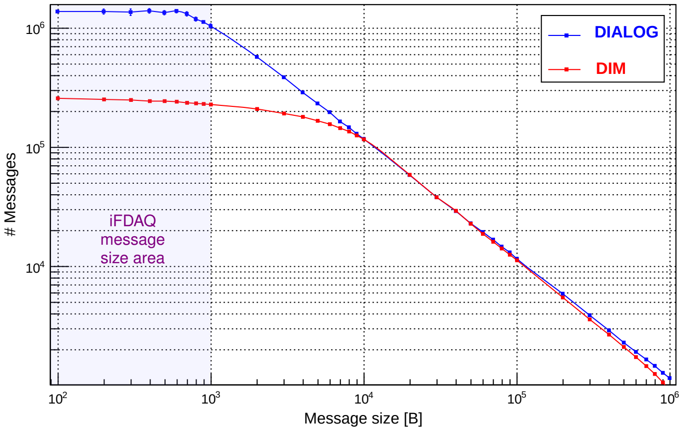


	Name of Process	Address	Port	PID	Provided services	Subscribed services	Commands
1	GUI	pccorc31.cern.ch	54214	4961	Show	Show	Show
2	Master	pccore15.cern.ch	48764	1033	Show	Show	Show
3	MSGBrowser	pccorc21.cern.ch	35666	24233	Show	Show	Show
4	MSGLogger	pccore15.cern.ch	58722	1034	Show	Show	Show
5	SC_RE11	pccore15.cern.ch	35647	1760	Show	Show	Show
6	SC_RE12	pccore15.cern.ch	35724	1900	Show	Show	Show
7	SC_RE13	pccore15.cern.ch	46410	2390	Show	Show	Show
8	SC_RE14	pccore15.cern.ch	42652	2320	Show	Show	Show
9	SMC01_RE11	pccore15.cern.ch	42630	1830	Show	Show	Show
10	SMC02_RE12	pccore15.cern.ch	47699	1970	Show	Show	Show
11	SMC03_RE13	pccore15.cern.ch	52851	2040	Show	Show	Show
12	SMC04_RE14	pccore15.cern.ch	41940	2110	Show	Show	Show
13	SMC05_RE15	pccore15.cern.ch	51730	2250	Show	Show	Show
14	SMC06_RE15	pccore15.cern.ch	58676	2460	Show	Show	Show
15	SR_RE11	pccore11.cern.ch	58103	22301	Show	Show	Show

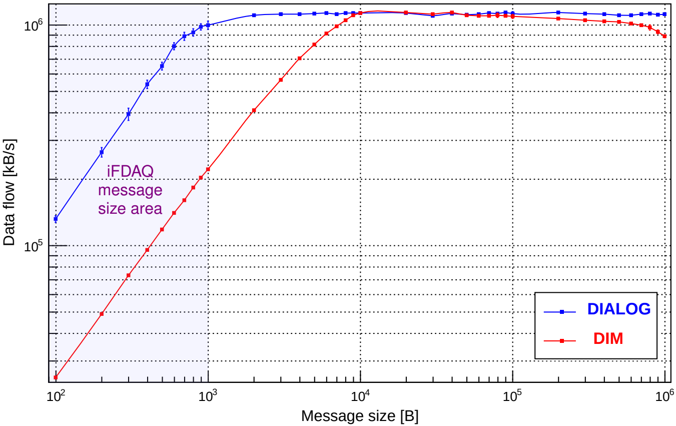
Performance Test I – Test Setup

- DIALOG and DIM performance are measured and compared in two plots
  - Number of messages – how many messages can be delivered to one single process in 1 second
  - Data flow – how many bytes can be delivered to one single process in 1 second
- 25 slaves (17 slave-control, 8 slave-readout) send status to Master process $\rightarrow$ the iFDAQ full setup
- The test is conducted for different message sizes and for each message size is conducted five times to obtain the sufficient statistics
- The network bandwidth is 10 Gbps for the test
  - We can expect the maximum data rate $\sim$ 1.2 GB/s (throughput)
  - The network bandwidth is not saturated by anything else during test
- Correct spreading of slaves among machines
  - Message sent by process 1 to process 2 running on the same machine is sent directly and it is not going through the network at all
    - The test results would have been even above the network bandwidth

Performance Test II – Number of Messages



Performance Test III – Data Flow



Conventional Debugging I

- Process of error detection within the program
  - Reproduce the problem
  - Isolating the source of the problem
  - Fixing the problem
  - Verification of the fix
- Problems of iFDAQ
  - Master process crashed several times per day in run 2014 and 2015
  - Slave-readout crashed many times per day in run 2014 and 2015
  - Processes crashed without any obvious reason or additional information

Conventional Debugging II

- Conventional debugging during the real data-taking
  - It would waste the beam time during crash investigation
  - The performance of debugged processes would be lower
  - The conventional debugging process would increase load on readout engine computers
  - The iFDAQ expert would have to be present 24x7 on site
- The conventional debugging is possible only during time without beam
- The errors do not occur without the real data-taking
- Conventional debugging is not usable and effective for the error detection

DAQ Debugger I

- Library for the iFDAQ error detection
- Fully incorporated to all processes during the run 2016 and 2017
- Design requirements
  - The integration to running system requires interface for an easy use
  - It does not affect the process performance
  - It does not increase load on readout engine computers
  - It provides with reports in `/tmp` folder containing stack trace of all threads and memory dump

DAQ Debugger II

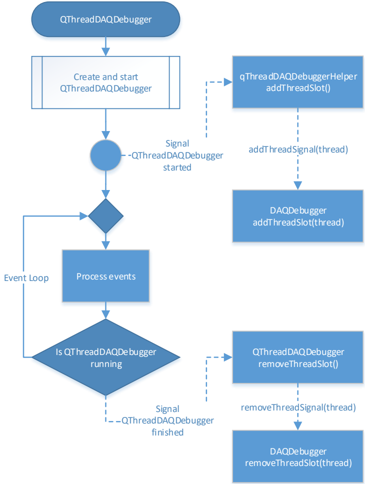
- Main goal is to produce a report of the process crash
- Based on catching of system signals (SIGSEGV, SIGABRT, etc.)
  - The system signal is caught and forwarded to a signal handler in the DAQ Debugger
  - The memory dump is produced and stored
  - The whole stack trace for each thread is generated with file names and code line numbers
  - The report containing the caught signal and stack trace for each thread is created in `/tmp` folder
  - The process is exiting with the caught signal
- Reports are investigated by iFDAQ experts
- Problem understanding → the fix is released and tested

DAQ Debugger III – Implementation

- `DAQDebugger::init(processName)` to initialize
- Crash procedure
  - The system signal is caught in the crashed thread
  - All remaining threads are immediately suspended
  - Store memory dump
  - Get stack trace of the crashed thread
  - Get stack traces of suspended threads
  - The crashed thread (whole process) is exiting with the caught signal
- Using POSIX defining a standard threading library API (suspend/resume signals)
- Using `backtrace`, `backtrace_symbols` and `addr2line` to create a report
- Using `gcore` for memory dump storage

DAQ Debugger IV – Thread Life Cycle

- The `QThreadDAQDebugger` object inheriting from `QThread` object
- To control thread via POSIX, thread ID is necessary to obtain using `QThreadDAQDebuggerHelper`
- Registration of thread in DAQ Debbugger by `addThreadSlot(thread)` method
- Standard thread execution with processing of events until the thread finishes
- `QThreadDAQDebugger` object finishes its execution
- Unregistration of thread in DAQ Debugger by `removeThreadSlot(thread)` method
- For simplicity reasons, the thread crash is not depicted in the diagram



DAQ Debugger V – Before Process Crash

- DAQ Debugger is a part of a process and standing in the background of a running process
- A process is running smoothly → the DAQ Debugger does not take any action → no effect on the process performance and no load increase on readout engines
- The system signals are registered, the process continues its execution
- Once the crash of process occurs, the DAQ Debugger handles it

DAQ Debugger VI – Process Crash

- Process crash $\rightarrow$ the system signal is emitted
- It is caught by the signal handler of crashed thread in the DAQ Debugger
- The crashed thread sends the suspend signal to all remaining threads
- The memory dump is produced and stored
- The report file is created and open for writing
- The crashed thread writes its stack trace to the file

DAQ Debugger VI – Process Crash

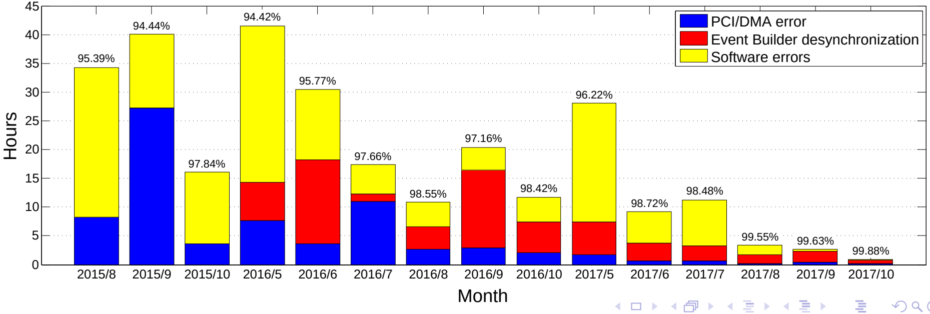
- The crashed thread sends the resume signal to first suspended thread and the crashed thread itself is suspended
- The resumed thread writes its stack trace to the file, then sends the resume signal to the crashed thread and is suspended again
- The resumed crashed thread sends the resume signal to second thread and it is again suspended
- The second resumed thread writes its stack trace to the file, then sends the resume signal to the crashed thread and is suspended again

DAQ Debugger VI – Process Crash

- It continues in this way to the last suspended thread
- The resumed crashed thread (resumed by the resume signal sent from $(n-1)$-th thread) sends the resume signal to n-th thread and it is again suspended
- The n-th resumed thread writes its stack trace to the file, then sends the resume signal to the crashed thread and is suspended again
- This suspend/resume procedure ensures the serial writing to file and proper thread control
- The report file is closed and process is exiting with the caught signal in the crashed thread

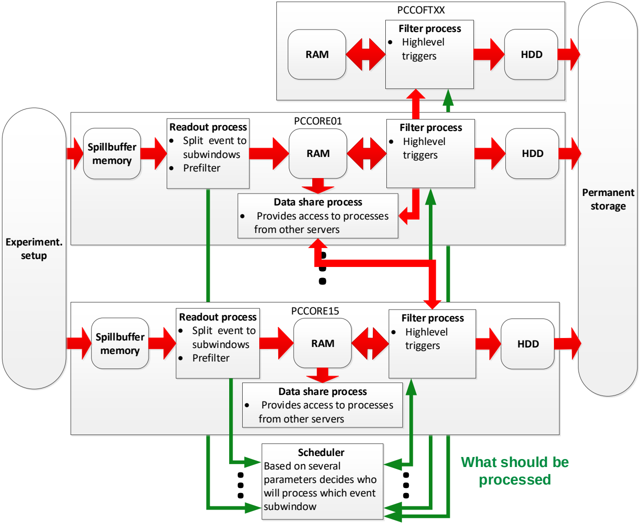
iFDAQ Stability

- iFDAQ Software is stable since October 2017
- Last observed iFDAQ Software crash is on 22nd September 2017
- DIALOG helped to increase stability of the iFDAQ
- DAQ Debugger detected all remaining software issues
- The iFDAQ UpTime – time when iFDAQ is able to take data



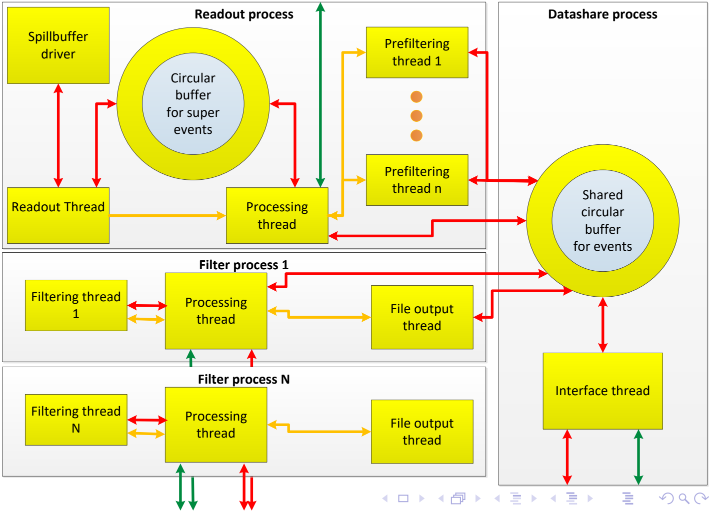
Overall data flow

- Readout process
  - Recieve events
  - Prefilter
- Scheduler process
  - Organize workload
- Data share process
  - Provides remote access to events in memory
- Filter process
  - High level trigger



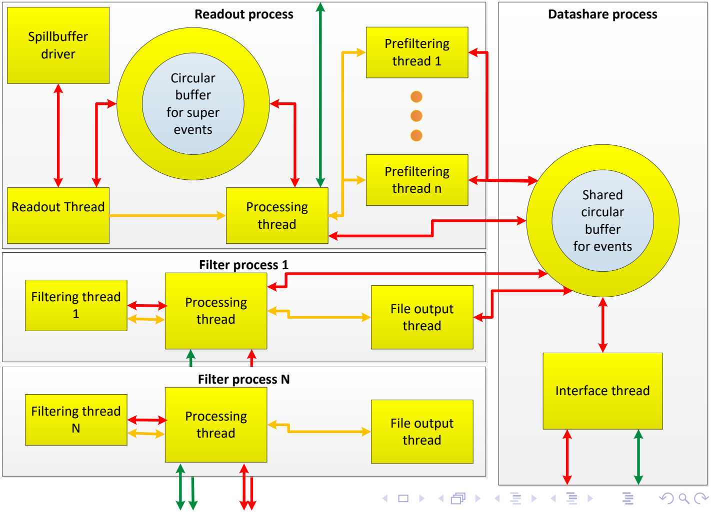
Detailed data flow - Readout process

- Gets data to inner memory as fast as possible
- Inner buffer for super events
- Several fast filter threads
- Info about event to scheduler



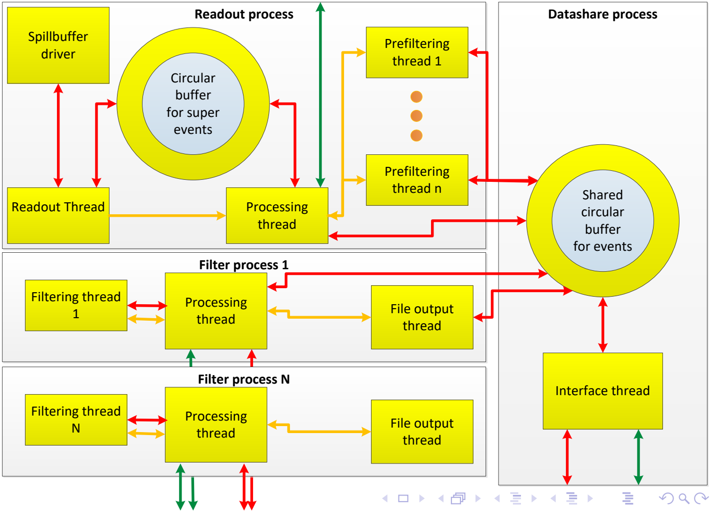
Detailed data flow - Datashare process

- Direct access to buffer of events in memory
- Exchange of info between filter process and scheduler
- Shares events from memory to requester



Detailed data flow - Filter process

- Gets commands from scheduler
- Gets events either directly from local memory or from Datashare process on different computer
- High level processing
- Stores filtered events on HDD



Conclusion

- Own library DIALOG
- Internal DAQ Debugger
- iFDAQ Software is stable since October 2017
- iFDAQ UpTime is around 99.88% $\simeq$ 1 hour loss / month of data-taking
- Continuously running DAQ
  - iFDAQ is running without starts and stops
  - It takes more data

The End

References

P. Abbon, et al.(the COMPASS collaboration): *The COMPASS experiment at CERN*. In: Nucl. Instrum. Methods Phys. Res., A 577, 3 (2007) pp. 455–518.

V. Y. Alexakhin, et al. (the COMPASS Collaboration): *COMPASS-II Proposal*. CERN-SPSC-2010-014, SPSC-P-340. May 2010.

M. Nakao , S. Y. Suzuki: *Network shared memory framework for the Belle data acquisition control system*. Real Time Conference, 1999. Santa Fe 1999. 11th IEEE NPSS.

C. Gaspar, M. Dönszelmann, Ph. Charpentier: *DIM, a Portable, Light Weight Package for Information Publishing, Data Transfer and Inter-process Communication*. International Conference on Computing in High Energy and Nuclear Physics, Padova, Italy, 1-11th February 2000.

C. Gaspar, M. Dönszelmann: *DIM – A Distributed Information Management System for the DELPHI Experiment at CERN*. Proceedings of the 8th Conference on Real-Time Computer applications in Nuclear, Particle and Plasma Physics, Vancouver, Canada, June 1993.

C. Gaspar, J. J. Schwarz: *A Highly Distributed Control System for a Large Scale Experiment*. 13th IFAC workshop on Distributed Computer Control Systems – DCCS'95, Toulouse, France, 27-29th September 1995.

M. Bodlak, et al.: *Development of new data acquisition system for COMPASS experiment*. Nuclear and Particle Physics Proceedings, 37th International Conference on High Energy Physics (ICHEP). April–June 2016, vol. 273–275, pp. 976–981. Available at: http://dx.doi.org/10.1016/j.nuclphysbps.2015.09.153.

M. Bodlak, et al.: *FPGA based data acquisition system for COMPASS experiment*. Journal of Physics: Conference Series. 2014-06-11, vol. 513, issue 1, s. 012029-. DOI: 10.1088/1742-6596/513/1/012029. Available at: http://stacks.iop.org/1742-6596/513/i=1/a=012029?key=crossref.78788d23de2b4a6a34d127c361123b8c.

M. Bodlak, et al.: *New data acquisition system for the COMPASS experiment*. Journal of Instrumentation. 2013-02-01, vol. 8, issue 02, C02009-C02009. DOI: 10.1088/1748-0221/8/02/C02009. Available at: http://stacks.iop.org/1748-0221/8/i=02/a=C02009?key=crossref.a76044facdf29d0fb21f9eefe3305aa5.

References

M. Bodlak, et al.: *Developing Control and Monitoring Software for the Data Acquisition System of the COMPASS Experiment at CERN*. Acta polytechnica: Scientific Journal of the Czech Technical University in Prague. Prague, CTU, 2013, issue 4. Available at: http://ctn.cvut.cz/ap/.

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

C. Ghabrous Larrea, et al.: *IPbus: a flexible Ethernet-based control system for xTCA hardware*, 2015 JINST 10 C02019. doi:10.1088/1748-0221/10/02/C02019.

CASTOR – CERN Advanced Storage manager. Available at: http://castor.web.cern.ch. (Accessed: 2017-05-01).

Electronic developments for COMPASS at Freiburg. Available at: http://hpfr02.physik.uni-freiburg.de/projects/compass/electronics/catch.html. (Accessed: 2017-05-01).

The GANDALF Module. (online). Available at: http://hpfr03.physik.uni-freiburg.de/gandalf/pages/information/about-gandalf.php?lang=EN. (Accessed: 2017-05-01).

iMUX/HGESICA module. (online). Available at: https://twiki.cern.ch/twiki/pub/Compass/Detectors/FrontEndElectronics/imux_manual.pdf. (Accessed: 2017-05-01).

Linux at CERN. (online). Available at: http://linux.web.cern.ch/linux/scientific6/. (Accessed: 2017-05-01).

S-Link – High Speed Interconnect. (online). Available at: http://hsi.web.cern.ch/HSI/s-link/. (Accessed: 2017-05-01).