PAUL SCHERRER INSTITUT

PSI

Michele Brambilla :: Scientific Software Developer :: Paul Scherrer Institut

Upgrade of SINQ histogram memory

BrightnESS meeting, Lund

PAUL SCHERRER INSTITUT

Outline

- Motivations
- Event generators
- SINQ histogram memory

Motivations

SINQ

- SINQ is the Swiss spallation neutron source
- continuos source wit a flux of about 10^{14} n/(cm^2 s)
- equipped with different instruments: diffractometers,small-angle scattering,reflectometers, spectrometers, …
- major shutdown & upgrades in winter
- some of the instruments are getting old, in particular electronics
- the instrument RITA2 is currently undergoing such an upgrade

PAUL SCHERRER INSTITUT

RITA2 instrument upgrade

RITA2 is a **triple-axis spectrometer** designed for moderate flux of cold neutrons combined with an extremely low background

- the **main detector** is an area sensitive detector with 128x128 pixels
- the electronics has been conceived in the ‘80s: it is getting quite old, there are no spare parts nor documentation…
- during SINQ shutdown RITA2 electronics has been upgraded
    - 2nd generation data acquisition
    - **event streaming** with 2nd generation DAQ

PAUL SCHERRER INSTITUT

In order to develop the histogram memory SW we need

- an event streaming format (BS-like)
- an event generator that provides data in event format

The idea beyond using an event generator

- read NeXus data file for the instrument
- convert into event format
- send to the histogram memory & histogram
- compare initial and final data
- recently the streaming interface for the electronics was completed, we can work with “the real thing”

Development strategy

We (still) have no real-time data: implementation was only possible thanks to the event generator.

It was also the playground to learn about 0MQ.

- **nEventGenerator**: C++, 0MQ, reads NeXus file. Data format similar to *bsr* wrt final implementation at PSI. Consists of a generator+reader
- **nEventGeneratorPy**: python, 0MQ, NeXus, twisted. Data & header format changed to agree with final RITA2 DAQ electronics specification. Generator+reader, can be driven from external signal via twisted
- **mcstasGenerator**: python, 0MQ, mcstas. Same data format of RITA2, can read the output of a mcstas simulation and stream data. The idea is to define 1D, 2D and N-D detectors and build instruments on top of them

Can be used not only to emulate data and explore bandwidth, but also to test hardware failures

mcstasGenerator

mcstas output consists of **ASCII files** for each detector in the instrument

Detectors (monitors) can be 1D (ToF), 2D (PSD) or n-D

- information on the simulation and values (min, max, content) introduced by “#”
- follow blocks of data: intensity, standard deviation, neutron counts

The idea is to “**build**” the **instrument**

- **detector** classes know how to read mcstas output (only 3 types are required)
- “build” the instrument class instantiating the detectors
- convert data into streaming format
- send data using 0MQ

Event data format


```
{
"htype":"sinq-1.0",
"pid":12345,
"st":1461844534.333,
"ts":3845255025,
"tr":100000,
"ds":[{"ts":32,"bsy":1,"cnt":1,"rok":1,"gat":1,"evt":4,"id1":12,"id0":12},30000],
"hws":{"error":0,"full":0,"zmqerr":0,"lost":[0,0,0,0,0,0,0,0,0,0]}
}
```


- “ds”: describe data format (64bit)
  - “ts”: event timestamp (32bit)
  - “bsy”, “cnt”, “rok”, “gat”: hardware status (1bit each)
  - “evt” : event type [channel id, position+channel, 2D] (4bit)
  - measured values *id0*, *id1* (12bit each)
  - number of events
- “hs”: status of streaming hardware
  - any error
  - buffer full
  - any error within 0MQ
  - counter for lost events

SINQ Histogram Memory

A few concerns as regards the software side of the upgrade:

- it ha to be performed during SINQ shutdown
- we have no time to write a brand new acquisition software
- the choice was to extend SINQhm to allow event streaming by making use of ØMQ

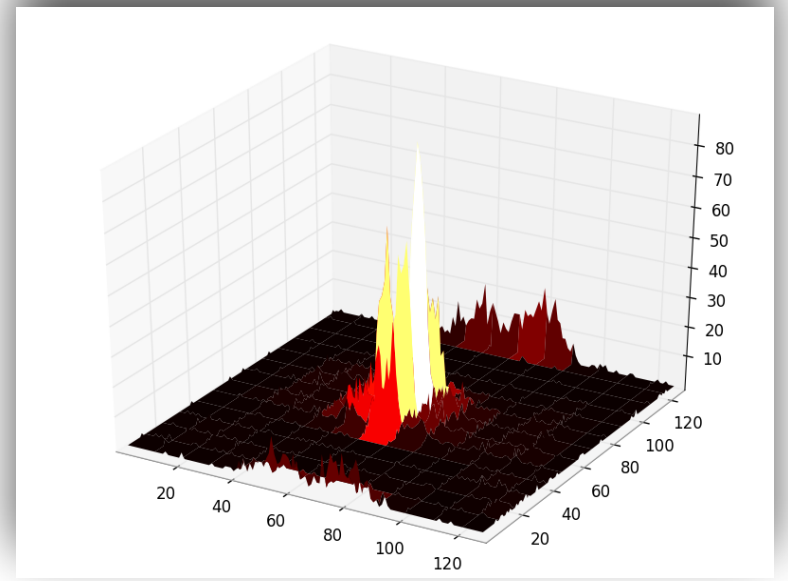
The SINQ histogram memory consists of two main parts:

- **histogram filler**: events acquisition and histogramming
- **EGI webserver** (users): monitor and control data acquisition

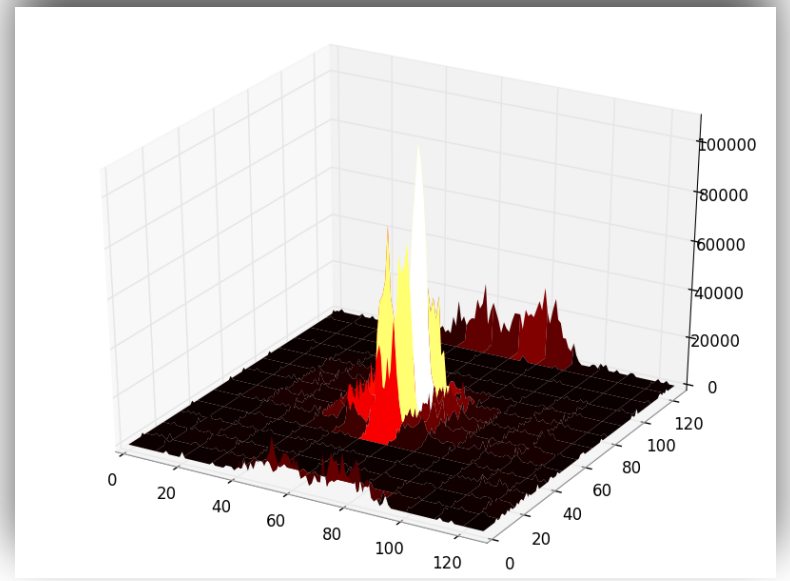
These are different process, communicate via **shared memory**

HM at work

original data



reconstructed histogram



difference in counts due to repeated streaming of same data

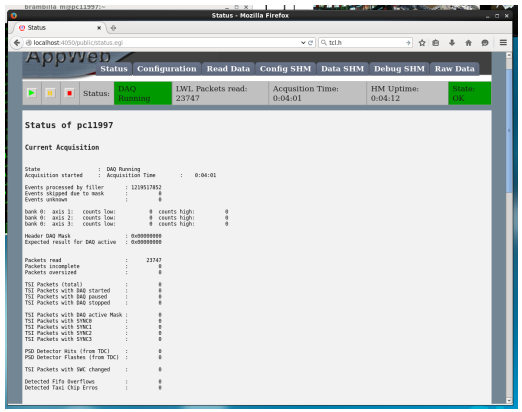
PAUL SCHERRER INSTITUT

PSI

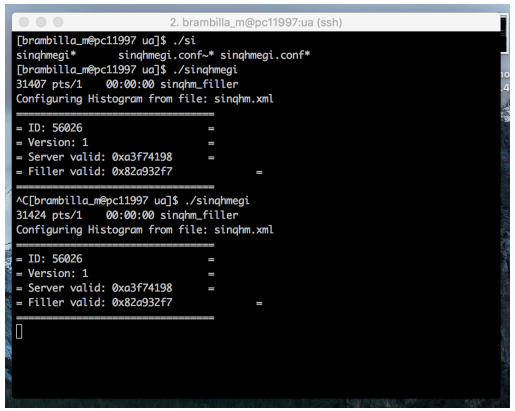
event generator

RITA2 electronics

sinqhmegi



shared memory



sinqhm_filler

download histogram

mongoDB

SICS monitoring

download raw data

(NeXus generation)

PAUL SCHERRER INSTITUT

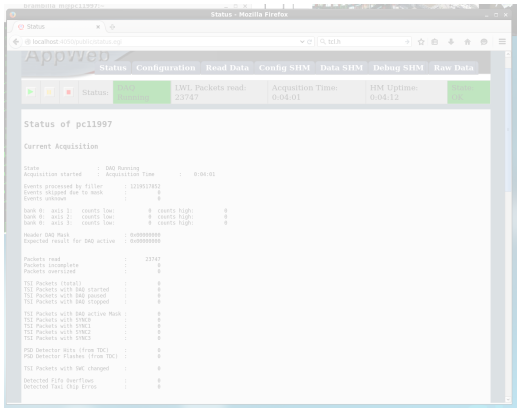
PSI

event generator

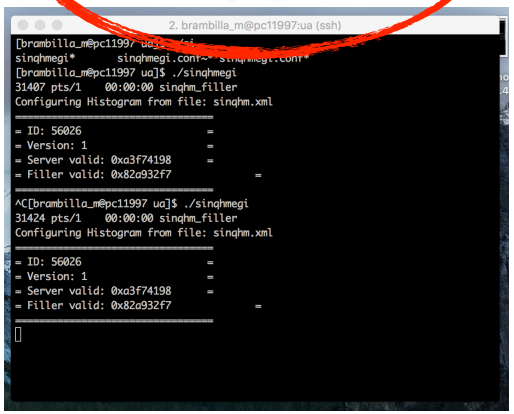
RITA2 electronics

Here's where ØMQ comes into play

sinqhmegi



shared memory



sinqhm_filler

download histogram

download raw data

mongoDB

SICS monitoring

(NeXus generation)

PAUL SCHERRER INSTITUT

Communications structure

```
extern volatile unsigned int *shm_cfg_ptr;

int zeromq_init()
{

    /* ... */
    if ( !shm_cfg_ptr[CFG_FIL_ZMQ_INI_DONE] ) {
        /* Creates 0MQ context and socket */
        /* if the latter fails notify */
        zmqContext = zmq_ctx_new();
        pullSocket = zmq_socket(zmqContext,ZMQ_PULL);
        if (pullSocket == NULL ) {
            shm_cfg_ptr[CFG_ZMQ_SICS_STATUS] =
ZMQ_CONNECT_ERROR;
            dbg_printf(DBGMSG_ERROR,"0MQ init failure\n");
        }
        /* ... */
    }

    status = zmq_connect(pullSocket,zeromq_bind_address);
    shm_cfg_ptr[CFG_FIL_ZMQ_INI_ERROR] = status;
    if ( status != 0) {
        shm_cfg_ptr[CFG_ZMQ_SICS_STATUS] = ZMQ_CONNECT_ERROR;
        dbg_printf(DBGMSG_ERROR,"0MQ connection error\n");
    }
    /* ... */
    status = zmq_setsockopt (pullSocket, ZMQ_TCP_KEEPALIVE,
                             &value, sizeof(value));

    /* ... */
    /* milliseconds before timeout */
    value = DEFAULT_TIMEOUT;
    status = zmq_setsockopt (pullSocket, ZMQ_RCVTIMEO,
                             &value, sizeof(value));

    shm_cfg_ptr[CFG_FIL_ZMQ_INI_DONE]  = 1;
    shm_cfg_ptr[CFG_FIL_ZMQ_FIRST_PKG] = 1;
    /* ... */
}
```

```
int zmqReceive(packet_type* p)
{

    if ( shm_cfg_ptr[CFG_FIL_ZMQ_INI_DONE] && ! shm_cfg_ptr[CFG_FIL_ZMQ_INI_ERROR] ) {

        /* header and data blod are sent with different zmq_send but as a single message (via
ZMQ_SNDMORE) */
        /* at the beginning ZMQ_RCVMORE flag is set to 0, so if this holds we are receiving the
header. */
        /* If the flag has value 1 we are receiving data without header! */
        zmq_getsockopt(pullSocket, ZMQ_RCVMORE, &rcvmore , &optlen);
        if (!rcvmore) {

            bytesRead =  zmq_recv (pullSocket, headerData, 1024, 0);

            if ( bytesRead < 0 ) {
                /* ... */
                /* Error: try reconnect */
                return ZMQ_RECV_TIMEOUT;
            }

            /* Get the flag value: if the message has another part, there will be data */
            zmq_getsockopt(pullSocket, ZMQ_RCVMORE, &rcvmore , &optlen);

            /* Parse header, if anything goes wrong notify */
            cJSON* root = cJSON_Parse(headerData);
            parse_header_value(root,"pid"       ,&(p->ptr)          );
            parse_header_value(root,"ts"        ,&tsCounter         );
            parse_header_array(root,"ds",""   ,1,&(p->length)      );
            if (p->length < 0 || p->ptr < 0 || tsCounter <= 0) {
                shm_cfg_ptr[CFG_ZMQ_SICS_STATUS] = ZMQ_HEADER_ERROR;
                shm_cfg_ptr[CFG_FIL_PKG_INCOMPLETE]++;
                dump_error("0MQ header warning",DBGMSG_WARNING);
            }
        }

        /* Start receiving data */
        while (rcvmore) {

            bytesRead =  zmq_recv (pullSocket, dataBuffer+bytesCnt, required_memory, 0);
            if ( bytesRead < 0 ) {
                /* ... */
                /* No data received before timeout, notify */
                return ZMQ_RECV_TIMEOUT;
            }
            /* Are there other parts? */
            zmq_getsockopt(pullSocket, ZMQ_RCVMORE, &rcvmore , &optlen);
            /* If the amount of received data is smaller than expected, notify */
            if ( bytesRead < required_memory && !rcvmore) {
                /* ... */
                return ZMQ_INCOMPLETE_PACKAGE;
            }

        }
        else {
            dump_error("Received data without header",DBGMSG_WARNING);
            shm_cfg_ptr[CFG_FIL_PKG_INCOMPLETE]++;
        }

    }
}
```

Data header + zmq_getsockopt::ZMQ_RCVMORE acts as **heartbeat**

- header will be sent even if data will not
- filler waits for data only if ZMQ_RCVMORE == 1

PAUL SCHERRER INSTITUT

DAQ structure

```
void config_zeromq_loop(void)
{

    shm_cfg_ptr[CFG_FIL_FILLER_STATE]=FILLER_STATE_CONFIG_LOOP;
    /**
     * on entering: CFG_SRV_DO_CFG_CMD == 0,
                    CFG_FIL_DO_CFG_ACK == 0,
     * if filler_valid == (nil) => keeps polling doing nothing
     **/
    while (1) {
        /* if user start/stop DAQ construct/destruct process and notify
*/
        if (shm_cfg_ptr[CFG_SRV_DO_CFG_CMD] && !
shm_cfg_ptr[CFG_FIL_DO_CFG_ACK]) {
            status = process_construct();
            /* ... */
            shm_cfg_ptr[CFG_FIL_DO_CFG_ACK] = 1;
        }
        if (!shm_cfg_ptr[CFG_SRV_DO_CFG_CMD] &&
shm_cfg_ptr[CFG_FIL_DO_CFG_ACK]) {
            process_destruct();
            shm_cfg_ptr[CFG_FIL_DO_CFG_ACK] = 0;
        }

        /* if the filler is valid, configuratin is done and DAQ is ON
*/
        if (shm_histo_ptr->filler_valid == DATASHM_CFG_FIL_VALID) {
            if (shm_cfg_ptr[CFG_SRV_DO_DAQ_CMD] &&
shm_cfg_ptr[CFG_FIL_DO_CFG_ACK]) {
                /* ... */
                daq_loop_0mq();

shm_cfg_ptr[CFG_FIL_FILLER_STATE]=FILLER_STATE_CONFIG_LOOP;
                shm_cfg_ptr[CFG_FIL_DO_DAQ_ACK] = 0;
            }
        }
        /* ... */
        /* Update stats & counters */
        update_pkg_stat_cnt();
        shm_cfg_ptr[CFG_FIL_ALIVE_CONFIG_LOOP]++;
        /* ... */
    }

}
```

```
void daq_loop_0mq()
{

    shm_cfg_ptr[CFG_FIL_FILLER_STATE]=FILLER_STATE_DAQ_LOOP;
    init_daq_zmq_start(&packet);  // zeros packet info & HM

    while ( shm_cfg_ptr[CFG_SRV_DO_DAQ_CMD] ) {

        if ( !shm_cfg_ptr[CFG_SRV_DAQ_PAUSE_CMD] )  {

            while (1) {

                status = zmqReceive(&packet); // receives zmq message &
update counters
                if (status != 0) {
                    packet.length = 0;
                    if ( /* ..._ZMQ_RCVTO] || !..._INI_DONE] || !
_ZMQ_INI_ERROR] */ ) {
                        /* ... */
                        zeromq_init();
                        continue;
                    }
                }
                if ( /* everything ok */ ) {
                    process_packet_fcn(packet);
                }

                /* regularly exit from loop and check ..._DO_DAQ_CMD */
            }

        } else {
            /* if ..._PAUSE_CMD receive without doing anything */
            /* regularly exit from loop and check ..._DO_DAQ_CMD */
        }


        /* alive counter & stats */
        shm_cfg_ptr[CFG_FIL_ALIVE_DAQ_LOOP]++;
        /* ... */
    }

    /* when _DO_DAQ_CMD] == 0 stop DAQ */
    leave_daq_loop();
}
```

Results and conclusion

Adapting the SINQ histogram memory to the upgraded RITA2 electronics was the opportunity to test future developments within BrightNESS

- ØMQ
- event generators

The final SINQhm solution satisfies our requirements

- stability
- recovering after disconnections
- bandwith (up to ~800* MB/s)

Expected to be “at work” in the next few days