Slice building system. Status and plans.

K. Gritsay on behalf of the SPD DAQ team.

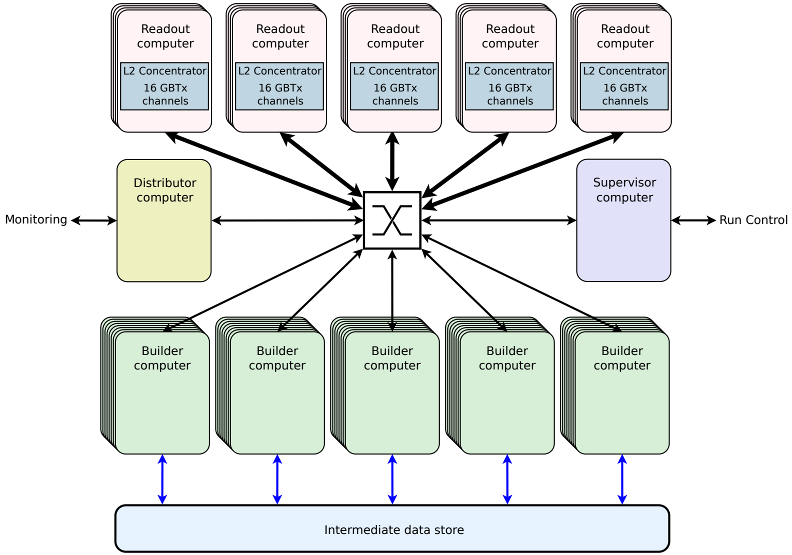
Joint Institute for Nuclear Research

24.10.2023

DAQ overview

- Data in the DAQ system is grouped by time into parts called slices. Slice length is around 10 μs. A sequence of slices forms a frame. Frame length is 1–10 s.
- The data is formed into slices and frames under the control of a time synchronization system that distributes the corresponding signals across all electronics modules.
- The reading chain, consisting of level 1 and 2 concentrators, transmits data to the readout computers. As a result, each readout computer receives its own parts of the slice data.
- A frame looks like a natural choice as a data processing unit, but in some cases frame size can be too large for convenient processing. Therefore the frame is programmatically divided into parts called chunks. A chunk is a data processing unit in the DAQ system that is transparent to the rest of the software. A reasonable chunk length is about 1 s.

Slice building system



Slice building system

Slice building tasks are:

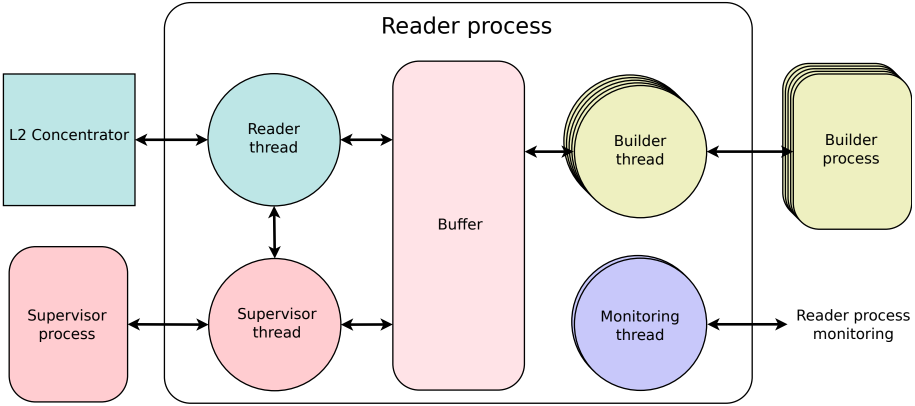
- Receiving data from L2 concentrators;
- Building a complete slice;
- Writing of the builded slices to the data storage;
- Distribution of some part of the builded slices for monitoring purposes across a group of computers.

The slice building system consists of following main processes:

- the **Reader** processes,
- the **Supervisor** process,
- the **Builder** processes,
- the **Distributer** processes.

To perform these processes, computers of four groups will be used: readout computers, supervisor computer, builder computers and distributor computers. The readout and supervisor computers will be located in Building 17 (SPD Building), and the builder and the master distributor computers will be located in Building 14 (NICA Data Center).

Reader process



Reader process

- The **Reader** receives data from the L2 concentrator, buffers the data in the RAM of the readout computer and sends the data in chunks to the **Builders** according to their requests.
- The **Reader** has a unique name directly associated with the ID of the L2 concentrator it serves.
- The **Reader** process is a multithreaded program. The main functional threads are:
    - The supervisor thread reads and executes a command from the **Supervisor** process.
    - The reader thread reads data from the L2 concentrator and stores it in the RAM buffer.
    - Builder threads send the requested data to **Builder** processes. For each **Builder** process, its own builder thread is created.
    - Monitoring threads send information about the **Reader** process to monitoring program. For each monitoring program, its own monitoring thread is created.

Reader process

- The **Reader** process is started as part of the system boot.
- At startup, it reads its base configuration from a file and forks the child process. After that, the parent process waits for the child process to finish. Everything else is done by the child process.
- The base configuration includes only name of the **Reader**, parameters for connecting to the L2 concentrator and opening of the listening sockets. It assumed that these parameters will not change frequently.
- The **Reader** process performs initial initialization and waits for an incoming connection from the **Supervisor**.
- As soon as the **Supervisor** connects to the **Reader** process, the **Reader** process receives the rest of its configuration from the **Supervisor**.

Reader process

- The parameters received from the **Supervisor** include chunk size, buffer size, and other informations that need to be changed consistently across all **Readers**.
- The **Reader** process completes initialization, starts the reader thread and waiting incoming connections from the **Builders**.
- After initialization, the **Reader** process reads and executes the regular **Supervisor** commands:
  - `RD_RUN_TEST`, `RD_RUN_START`, `RD_RUN_STOP`: checks whether the run can be started, starts or stops the run, respectively.
  - `RD_GET_MDATA`: gets metadata. In response of it, the **Reader** process sends information to the **Supervisor** about read chunks and slices, and also about chunks and slices already passed to the **Builders**, the buffer usage, and so on.
  - `RD_CHUNK_DROP`: specified chunks will be deleted from buffer.
- In case if connection to the **Supervisor** is lost, the child process is terminating and parent process forks another child process.

Reader process

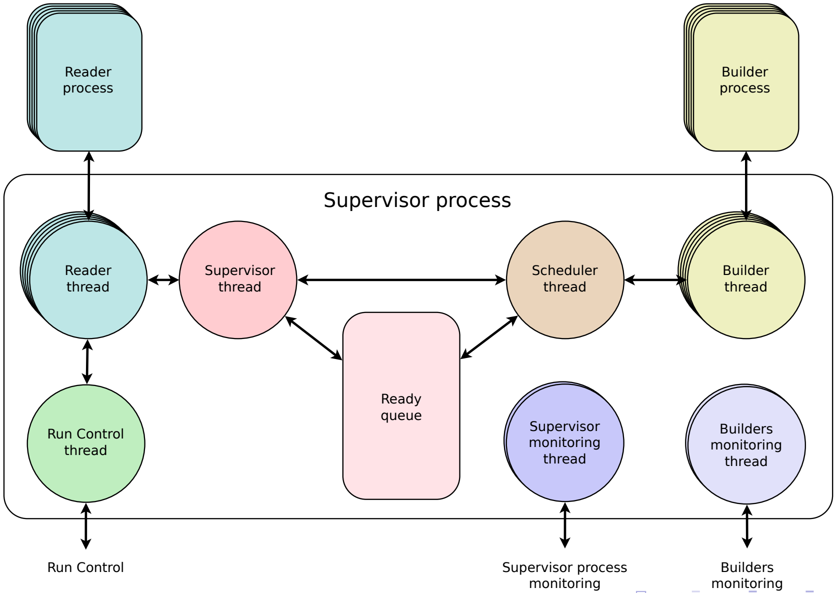
The reader thread.

- At startup, the reader thread connects to the L2 concentrator.
- It reads data in slices from the L2 concentrator, stores the data in the buffer, and updates the metadata. The data is addressed by the run ID received from **Supervisor** at the start of the run, and the chunk number, which is derived from the frame number and slice number received from the L2 concentrator.
- In case of an error during data reading, it closes the connection to the L2 concentrator and tries to reopen it.

The builder thread.

- At startup, the builder thread gets the required run ID and chunk number from the **Builder**.
- It sends the chunk data to **Builder** and releases the transferred slices from the buffer.
- The builder thread terminates when the transfer is completed or in case of an error. Unsent slices are released from the buffer.

Supervisor process



Supervisor process

- The **Supervisor** process collects metadata from **Reader** processes and chunk processing requests from **Builder** processes. Then it decides which **Builder** should process which chunk.
- The main threads of the **Supervisor** process are:
  - The run control thread reads and executes commands from the **Run Control**.
  - The reader and builder threads communicate with the **Reader** and **Builder** processes, respectively.
  - The supervisor thread relies on metadata from **Readers**, decides on chunks, and maintains a queue of chunks ready for processing.
  - The scheduler thread assigns chunks from ready queue to be processed by the selected **Builders**.
  - Two types of monitoring threads send information about the **Supervisor** itself and the connected **Builders** to the corresponding monitoring program.

Supervisor process

- At startup, the **Supervisor** process reads its base configuration from a file and forks the child process. After that, the parent process waits for the child process to finish. Everything else is done by the child process.
- The base configuration contains the names and addresses of configured **Readers**, addresses and port numbers of listening sockets, a description of the network topology and database connection parameters. It was assumed that these parameters would not change often.
- The child process reads the rest of the configuration from the database. This includes parameters passed to **Readers** and **Builders**, as well as other parameters that may be updated more frequently.
- The **Supervisor** process performs initialization and starts persistent threads.

Supervisor process

Run control thread.

- At startup, the run control thread waits for a connection from the **Run Control**.
- When the connection is established, the run control thread executes the received commands:
  - `SBS_GET_CONF`: returns a list of configured **Readers**.
  - `SBS_GET_STATUS`: returns the current status, which includes the run status, data queue flag, run number, run ID, data recording flag, and statuses of all configured **Readers**.
  - `SBS_READER_ON/SBS_READER_OFF`: enables/disables a **Reader**. If the **Reader** is disabled, it will be ignored by DAQ.
  - `SBS_DATA_QUEUE`: sets data queue flag. If the flag is off, then all chunks will be dropped.

Supervisor process

Run control thread (continue).

- commands (continue).
  - `SBS_RUN_TEST`, `SBS_RUN_START`, `SBS_RUN_STOP`: checking whether the run can be started, starting and stopping the run, respectively. As the parameters of the `SBS_RUN_START` command, the **Supervisor** receives the run number and the data recording flag. When starting a run, the **Supervisor** generates a unique monotonically increasing run ID.
  - `SBS_RESTART`: This terminates the **Supervisor's** child process, resulting in the termination of the child processes of all **Readers** and **Builders**, which in turn leads to the restart of the slice building system.
- In case if connection to the **Run Control** is lost, the run control thread waits for a new connection.

Supervisor process

Supervisor thread.

- It periodically reads metadata from all **Readers**.
- It deletes chunks with lost slices.
- It maintains a queue of chunks ready for processing by **Builders** (ready queue), leaves only the specified number of chunks in the queue and removes excessive chunks. The presence of excessive chunks indicates an insufficient number of **Builders**. Older chunks are deleted first.
- It wakes up the scheduler thread when there are chunks in the ready queue.

Reader threads.

- A single reader thread is created for each configured **Reader** process.
- The reader thread receives commands from the run control and supervisor threads and passes them to the **Reader** process, reads its responses. Communication with all **Readers** is carried out in parallel.

Supervisor process

Builder thread.

- Each connection from the **Builder** process creates one builder thread.
- When the builder thread starts, it sends configuration to the **Builder** process, which in turn sends part of its configuration to the **Supervisor** process. The parameters received from **Builder** include the **Builder** name, the **Builder** network location, and the connection bandwidth. The **Builder** name is not required to be unique. The parameters sent to **Builder** include the unique builder ID and the addresses of all configured **Readers**.
- After initialization is completed, the builder thread waits for the `SV_CHUNK_REQUEST` command from **Builder**, it informs that **Builder** is ready to process the chunk. The builder thread wakes up the scheduler thread and waits for its solution.

Supervisor process

Builder thread (continue).

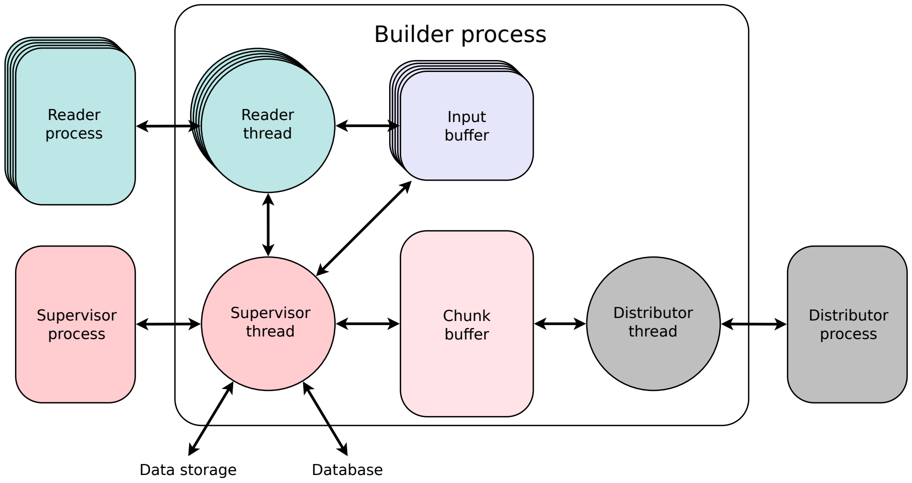
- When **Builder** is selected for processing, the builder thread sends it the command `BD_CHUN_ASSIGN` with a list of enabled **Readers**, the run ID and the chunk number assigned for processing.
- When **Builder** receives data from all **Readers** it reports this to the **Supervisor** by sending the command `SV_TRANS_FINISHED`.
- When **Builder** completes the chunk collection, it transfers chunk to the data storage and updates the database. After notifies about it to the **Supervisor** by sending the command `SV_BUILD_FINISHED`.
- The builder thread terminates in case of any communication errors with the **Builder** process.

Supervisor process

Scheduler thread.

- If the ready queue is not empty and there is at least one request for chunk processing from **Builders**, then either the supervisor thread or the builder thread wakes up the scheduler thread.
- The scheduler thread selects which **Builder** will process the chunk, and reports this to the corresponding builder thread.
- When choosing **Builder**, the order of receipt of processing requests and network load is taken into account. To do this, the **Supervisor** configuration has rather primitive means of describing the network topology, and the **Builder** configuration determines where and at what speed a particular **Builder** is connected. The scheduler thread selects **Builder**, which will have the highest network connection speed at the time of making the decision. Network load accounting can be useful if more than one **Builder** is running on the same computer.

Builder process



Builder process

- **Builder** reads data belonging to one chunk from all **Readers**, collects a complete chunk, writes it to the data storage and makes the corresponding entry in the database.
- In parallel with writing to the data storage, the **Builder** sends part of the chunk slices to the master **Distributor**.
- The **Builder** is multithreaded process, the main functional threads are:
  - The supervisor thread interacts with the **Supervisor** process, creates a complete chunk using data from input buffers, writes the created chunk to the data storage and updates the database.
  - The reader thread reads data from the **Reader** process and puts it into the input buffer. One reader thread is created for each **Reader** process and for each chunk being processed.
  - The distributor thread transfers the selected slices from the created chunk to the master **Distributor** process.

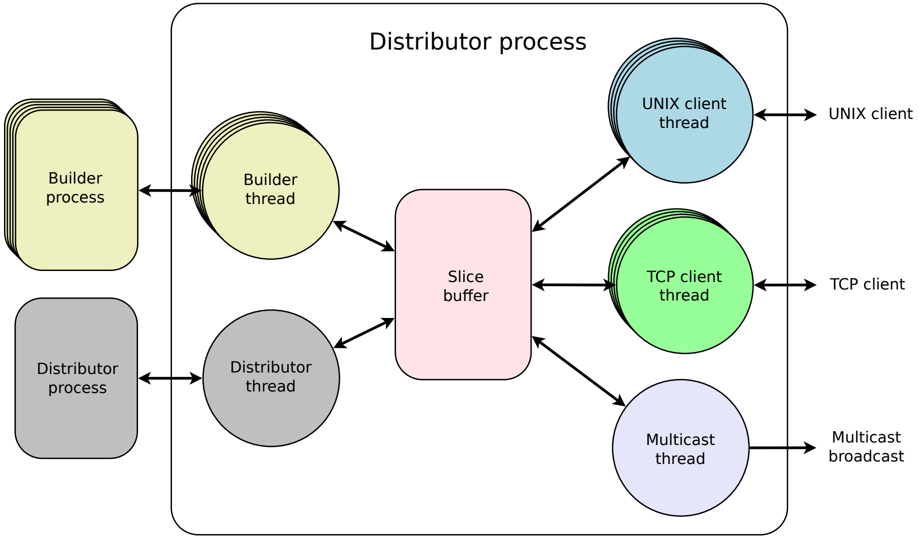
Builder process

- At startup, **Builder** reads its configuration from the file and forks the child process. After that, the parent process waits for the child process to finish. Everything else is done by the child process.
- The configuration includes name of the **Builder** and the addresses **Supervisor** and **Distributor**, as well as a description of the **Builder** connection to the network, including the connection speed.
- The **Builder** does initialization and starts persistent threads.
- The **Builder** connects to the **Supervisor**, sends **Builder** name and network connection parameters. From **Supervisor**, it receives a unique ID and addresses of all configured **Readers**.
- When **Builder** is ready to process data, it sends the `SV_CHUNK_REQUEST` command to **Supervisor** and waits for its response.

Builder process

- The **Supervisor** responds with the command BD_CHUNK_ASSIGN, containing the number of the chunk being processed, the data recording flag, and other information necessary for processing this chunk.
- The **Builder** creates reader threads, one for each **Reader** process. The reader threads receive data from all **Reader** processes in parallel and put them into input buffers. When the data transfer from all **Reader** is completed, the **Builder** informs **Supervisor** about it with the command SV_TRANS_FINISHED.
- The **Builder** creates chunk data using data from input buffers.
- The **Builder** wakes up the distributor thread and writes the chunk to the data storage, then makes the appropriate entry to the database. After that, it informs the **Supervisor** with the SV_BUILD_FINISHED command.
- If the connection to the **Supervisor** is lost, the child process terminates and the parent process forks another child process.

Distributor process



Distributor process

- **Distributor** collects data from **Builders** and distributes them to a group of computers for the purpose of data monitoring.
- Each **Builder** sends the specified number of slices from each chunk that it processes to **Distributor**.
- The received slices will be stored in a memory buffer.
- The slices are ordered by the run ID, frame number, and slice number.
- It is impossible to guarantee the correct order of slices received from different **Builders** but, at the same time it is necessary to ensure the correct order of slices for clients. So, clients will receive slices with a certain delay, which makes it possible to insert a slice with a smaller number into the buffer before a slice with a larger number but received earlier.

Distributor process

- Slices will be saved in the buffer for some time, which will allow clients to get some freedom of action. Old slices are removed from the buffer.
- The client can read the slice only once by requesting (explicitly or implicitly) the “next” slice, a request for a specific slice is not provided. The **Distributor** always returns the oldest unread slice presented in the buffer.
- The following methods are supported for distributing slices to clients:
  - The client reads the slices one by one, executing a read request for each slice. Both TCP and UNIX sockets are supported.
  - The client opens a connection and receives a stream of slices without requesting each slice. Both TCP and UNIX sockets are supported.
  - Slices are broadcast using UDP multicast.
- The **Distributors** can organize in chain. In this case secondary **Distributor** can connect to master **Distributor** using TCP (in stream mode) or UDP multicast.

Current status and plans

- The **Reader**. With the exception of one very important thing — interaction with the L2 concentrator, the **Reader** is in good readiness for testing within the system. For test purposes, the **Reader** can get data from an external data generator or generate empty data itself.
- The **Supervisor**. In general, it is in good readiness for testing as part of the system. There is no interface with the database yet.
- The **Builder**. Currently, **Builder** does not create chunks, does not write chunk data to data storage, does not interact with the database, and does not support interaction with the **Distributor**. It just implements interaction with the **Supervisor** and **Readers**, including the transmitting of simulated data. Its readiness is sufficient for testing within the system.
- The **Distributor**. Currently, realized interaction with the Builders, buffering of the slices and TCP connection for for clients.

Current status and plans

- Test bed for DAQ and online filter. We ordered 24x10 Gbps network switch and 5 computers with 2×10 Gbps interfaces and 256 GB memory. Together with 2 existing computers, the ordered equipment will be used to create a test bed.
- New computers are ordered in a configuration suitable for the upcoming work with the L2 concentrator.
- The immediate goal is to test and debug the slice building system software.
- Writing missing components that can be tested without an L2 concentrator.
- One of the long-term goals should be to clarify the parameters of the equipment for DAQ system.