The ToolDAQ DAQ Software Framework & Its Use In The Hyper-K & ANNIE Detectors

Benjamin Richards[1,*]

[1]Queen Mary University, London UK

Abstract. DAQ software is often a little-considered part of an experiment, with many experiments either using a rigid, heavily convoluted multi experiment framework or custom built software that lacks features and is fragmented. Equally these solutions tend not to be fault tolerant, scalable, distributed, dynamic to hardware changes or easy to produce. Presented here is ToolDAQ a C++ DAQ framework which has been designed to be easy and fast to develop for in a modular and simple way. It has many features such as in-built service discovery, dynamic reconfiguration, remote control/monitoring via web and terminal interfaces and a highly scalable fault-tolerant network communication infrastructure provided by ZeroMQ (ZMQ) built in. It is also compatible with newer and older hardware, both being very lightweight and with low dependencies. The framework is currently in use on the Accelerator Neutrino Neutron Interaction Experiment (ANNIE) in Fermilab and has been used to develop the DAQ for Hyper Kamiokande (Hyper-K) , whose implementations and use cases we will discuss, as well as many other places like stand-alone hardware and the intermediate water Cherenkov detector (E61) at J-PARC.

1 Introduction

A lot of high energy physics research and development is spent on designing the next generation of hardware components required to run a detector, however less time is spent on the DAQ software that controls them and records the data. In this area the level of innovation falls behind that of other world leading aspects, mainly due to lack of manpower, resources and knowledge. This leaves most experiments with a few normal avenues, either development of their own custom DAQ software, which tends to be the minimum required at the detector turn-on, with bugs worked out and features added over time, or code from another experiment is used and adapted. Both of these solutions, whilst possible, are not conducive to good structure and performance of this code and rarely have high levels of fault tolerance. A couple of general purpose frameworks do exist, however most tend to be either heavily bloated or convoluted, with many dependencies and rigid in their structure and a large barrier for entry. Added to this, none to my knowledge have a high level of fault tolerance. To address this a new general purpose framework was created called ToolDAQ [1]. It was constructed from scratch to address each of the above issues, so the structure would be correct and flexible enough at conception. It was also motivated by the need to get brand new experiments up and running in a very short timescale.

[*]e-mail: b.richards@qmul.ac.uk

ToolDAQ was designed from the ground up to provide flexibility and ease of use, whilst being as lightweight as possible with low dependencies in pure C++. It was designed to use standard software design paradigms to help structure and organise DAQ in a modular way that was scalable. This scalability was of particular importance as it was designed with the intent of it being used for simple hardware test stands right up to 3 different international particle physics detectors, each an order of magnitude bigger in size that the last. The modularity aspect of this code was also very important as component parts could be developed in test stands and then pushed out to multiple experiments where common components could be reused between them. The final consideration when designing the software was that fault tolerance / correction and dynamic routing of data were of high importance as the experiments it was intended to be used in would run for many years, even decades, and with critical data where losing even a single event could have large consequences.

2 Structure

To achieve this the program makes use of a pipeline model of execution which is referred to as a ToolChain (Figure 1). ToolChains are populated dynamically using factories with modular class component blocks known as Tools. The factories allow for dynamic reconfiguration of the ToolChains without re-compilation and this is achieved via use of easily editable ASCII configuration files in a simple format. These files are read into the software using a specialised storage class known as a Store. This Store is effectively a map of key objects to a non typed value. The type is defined at archiving and retrieval by use of templates. It also has IO capabilities supporting a simple ASCII format flat JSON tree and a binary data format. Store objects can also be streamed in code and have the ability to store and retrieve multiple instances of themselves creating multi-event object based storage. Custom classes can be stored in a store with a little modification to their code and all standard types and STL containers as well as pointers are supported.

This allows an easily understandable interface for people to interact with the code dynamically without any prior knowledge of a language or format. Each modular Tool class can also load its own dynamic configuration variables via use of a store instance and its own ASCII configuration file, for use in the Tool. This system nicely modularises configuration settings in accordance with the modularisation of the tools allowing for easy location of the configuration variables required.

The production of Tools and a ToolChain's configuration files are all handled by bash scripts to reduce any inheritance or formatting issues from the user. The makefile used to compile the software will automatically know about any newly added tools as the script that generates them adds them both to the factories and lists them in a unity file for automatic compilation. As mentioned previously ToolChain's configurations are generated by a script and are dynamic, so no recompilation is required if any number or order of Tools are added reordered or removed, or if new configurations are made. The Tools themselves ensure encapsulation and good practices in terms of the transparent handling of data in a Tool-independent way, as they cannot communicate / pass information directly between one another. Instead local data objects are private to that Tool and shared data objects between Tools are stored in a transient data model class known as the data model. This class was designed to be entirely free of restrictions to allow each implementation to decide how to handle and organise their data and what protection or enforcement of structure to provide. A default map of Store containers that was mentioned above is provided for people to fill out these if they would like a dynamic data storage object. Custom classes can be used to extend this data model which if placed in the same directory with it and included in the data model will be automatically compiled with the code and distributed via include statements for any Tool to use.

https://doi.org/10.1051/epjconf/201921401022

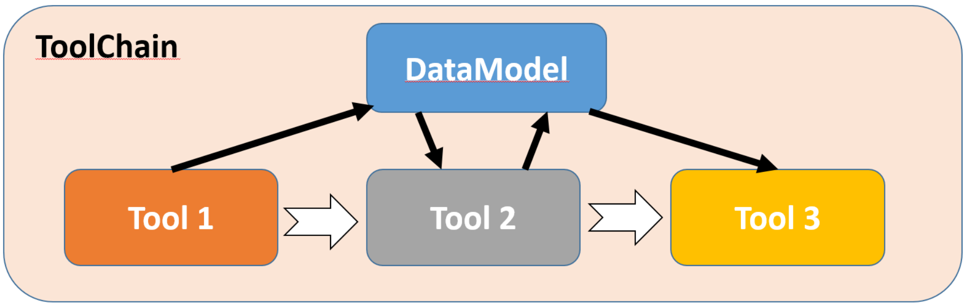

Figure 1. ToolDAQ's ToolChain pipeline showing modular Tool blocks and transient DataModel

3 Execution

In terms of operation of the software there are three running modes, interactive, inline and remote. The interactive creates a new thread which allows for command line execution, starting and stopping status of the Toolchain. Inline allows for a fixed number of executions before program exit with the option to do infinite number and be stopped by user condition. Remote allows for the same commands as interactive to be provided by remote TCP/IP connection in a separate thread via a remote management software which is discussed later. The actual execution of the Tools in the Toolchain via all three modes happens in three steps, initialise, execute and finalise. These steps execute the member functions of the Tools with the same names and can be run automatically with a start and stop command to the ToolChain or individually for greater control and debugging. These commands will then be run sequentially over the Tools within the Toolchain in the order with which they were added. The start command will initialise all Tools sequentially and start an indefinite loop of execute calls until a stop is called whereby the execute loop is stopped and the finalise is called. Tools themselves can have multiple threads running inside them allowing for parallelisation of tasks and can even contain a Toolchain within a Tool, meaning that modular tool classes can be subdivided further into more modular components. The execution of this sub ToolChain is then independent. The two can also be combined allowing a ToolChain to be run on separate threads within a single tool.

4 Building Fault Tolerance, Scalability & Dynamic Service Discovery

Most of ToolDAQ's network layers make use of message queuing via ZeroMQ (ZMQ) [2]. This external dependency allows for multiple features to exist. Firstly the included pair based socket type is used for all threads within the software. This can be tens of threads just in the framework itself and due to this inter-thread communication paradigm the threads can be completely thread safe without the need for mutex locks. The other benefit is that most of the external sockets connections both by the user and ToolDAQ can make use of message queuing and N-N based scalability as well as some very good fault tolerance behaviour which is greater than that of normal sockets. These tools can be used to set up multiple dynamic connections and used to provide fault tolerance of those connections between computers but there is a missing piece that is required for true automated re-routing of data. That piece is service discovery. This is achieved by every instance of a ToolChain on a network transmitting threaded periodic multicast beacons. These contain identification details, status details and remote connection details. The status transmitted can be adapted by users to inform about the current health of a ToolChain and extra open connections that other services from embedded Tools that others can connect to. Each ToolChain also has a thread responsible for maintaining a list of all the other broadcasting ToolChains on the network. This collates and organises the multicast beacons to give a picture of which machines exist, what is their status, purpose and how they can be connected to. Use of this with the above messaging paradigms allows for fully dynamic routing to occur. This means that the system can not only be fault tolerant but fault correcting. It also allows for a selection of tools for monitoring logging and remote control of the ToolChains. This is how remote control is achieved and a program designed for it is available that from any node on the network will collect the multicast beacons and then allow commands to be sent to any that are in remote mode. These messages are sent via normal TCP/IP using ZMQ and are structured in a flat JSON format.

5 Node Organisation

Due to the fact that remote nodes may be inaccessible, able to crash, remotely operated and numerous, a system using a daemon was created for node management, pushing new code out and organising the processes on that node. This Node daemon process runs in the background and broadcasts multicast beacons about the node's health. This lightweight program allows remote operators to see the node's state and issue commands to start ToolChain processes. If these ToolChains become unresponsive it also allows them to be closed forcibly and even the node rebooted for management. This all occurs again using TCP/IP and multicast beacons with ZMQ handling the message queues for the TCP/IP communications.

6 Monitoring & Control

Another nice feature of using ZMQ is the plethora of bindings available in multiple languages, meaning applications can be written in many languages to interface with the ToolDAQ system. One such important area is that of control and monitoring. A C++ terminal application for remote control exists as previously mentioned, however using these bindings, web interfaces to the monitoring data streams and ToolChain beacons and control have been created. These allow for simple direct access to hardware so experiments can design their tools to easily interface with ToolDAQ and expand on the provided templates.

7 Experimental Deployments

The ToolDAQ framework was first deployed on hardware test stands in the UK for waveform electronics and PMT sensor testing. This expanded to a deployment of the ANNIE experiment and is now used for the upcoming Hyper Kamiokande [3], both of which are described in detail below. The intermediate water Cherenkov experiment on the same beamline Hyper-K known as E61 is also developing their DAQ with it [4]. Other neutrino physics experiments have also indicated their interest in upgrading their systems to run with ToolDAQ and standalone calibration devices are being developed to use it. The ToolDAQ Framework's reach has now even expanded beyond that of DAQ. The ANNIE experiment now uses ToolDAQ for data post processing, monitoring, reconstruction and analysis, as well as their DAQ [5]. The Framework has also been expanded to accept Python based Tools. The Super Kamiokande experiment is developing GPU based online trigger systems with it and a version is now publicly available as a companion to simulation software for trigger development. The reason for this is because the pipeline based modular structure with transient data storage can be used for any application, DAQ or otherwise, and the flexible framework allows for fast efficient modular development with the specific DAQ aspects easily turned off.

7.1 Accelerator Neutrino Neutron Interaction Experiment (ANNIE)

The ANNIE experiment has been a great test bed and proving ground for the ToolDAQ software marking its first deployment on a real active detector. The experiment, which measures neutrino neutron interactions, was constructed in a very short timescale so the DAQ software was created from scratch in a couple of months. Also as the detector is a test experiment, so large numbers of major changes to the hardware and triggering were undertaken during its first phase of operation and the software had to be flexible enough to adapt to this. Another interesting feature of this experiment is that its 3 detector subsystems all have entirely asynchronous data streams, work on wildly different hardware standards and ages and all have to be controlled, synced and processed by the DAQ software, which really allowed thorough testing of the flexibility of the framework and its fault tolerance. The detector is also undergoing an order of magnitude increase in the photo sensors and other systems plus the addition of a 4th asynchronous stream for novel LAPPD hardware for its Phase 2 due to start in early 2019, so the scalability of the system will also be tested.

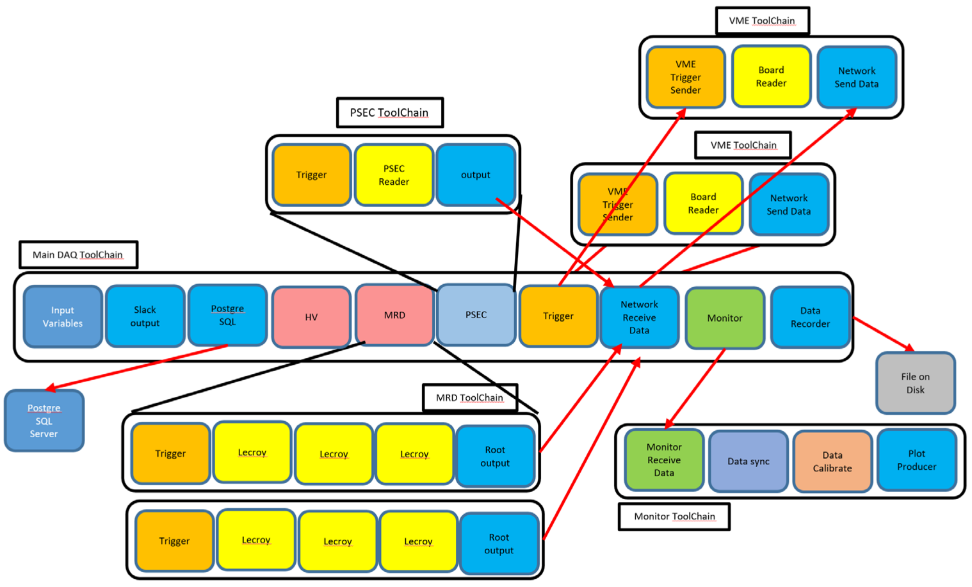

Figure 2. ANNIE's ToolDAQ based DAQ for Phase 2

The structure of ANNIE's Phase 2 DAQ is shown in Figure 2. It consists of a main ToolChain with Tool components for run settings, database synchronisation management and entry, trigger distribution across multiple nodes running their own ToolChains (some on embeded VME CPUs) and fault tolerant data transfer blocks that buffer data at both ends. The MRD CAMAC based TDC system and the LAPPD electronics interface run as threaded asynchronous ToolChains within Tools that send their data streams for collation upon request. Other tools exist for the purpose of monitoring and calibration. The health and status of every component of this system is monitored and controlled via node daemons and a web interface as previously described.

7.2 Hyper Kamiokande (Hyper-K)

The Hyper-K DAQ design using ToolDAQ (Figure 3) is a lot larger and more distributed. Some 120 server nodes will each run one or multiple ToolChains and are involved with reading out the ≈2,300 front end electronics boards (FEEs) which connect to the ≈55,000 PMTs in the water volume. These FEEs will be read out using TCP/IP by ≈ 80 readout buffer units (RBUs) via network switch hardware. These RBUs contain heavily threaded ToolChains for organising and buffering the data in a dead timeless way. The data is time sliced and then sent through various ZMQ socket types using a broker assisted paradigm to trigger processing units (TPUs) and event builder units (EBUs). Fault tolerance and resubmisison of time slice jobs is handled by the Brokers that have a master slave dynamic redundancy and which maintains dynamic queues of jobs and available nodes. This allows for hot swapping of active nodes due to the service discovery of ToolDAQ. Also the message queues on each node are buffered both at sender and receiver to provide greater fault tolerance. The TPUs worker nodes make use of GPUs for accelerated trigger decisions allowing for real time reconstruction of low energy events. These GPU trigger algorithms are fully integrated into the ToolDAQ framework and a separate ToolDAQ application has been created for their development and use on simulated data.

The intention for the system is to be infinitely scaleable and almost entirely self maintaining and correcting with data rerouted and automatically buffered so no loss occurs with hardware failure. Detector node components can be increased dynamically to upgrade resources thanks to the hot swapability allowing adding on RBUs, TPUs, EBUs and brokers as needed. These components will be sensed by other nodes on the system and joined to the system automatically. The run control and monitoring of the system will take place via a web browser with each node reporting its health and current status.

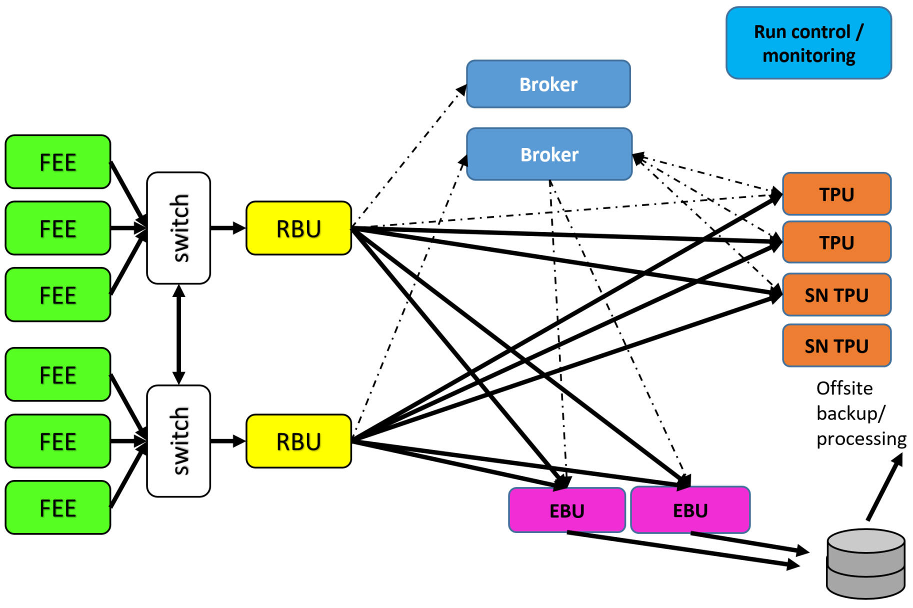

Figure 3. Hyper-K's DAQ schematic showing distributed node topology of the detector readout system. Nodes shown include the front end electronics (FEE), readout buffer units (RBU), trigger processing units (TPU), event builder units (EBU) and management brokers

8 Conclusion

The ToolDAQ framework is an open-source lightweight C++ DAQ framework, that is highly modular in structure. It makes use of ZMQ to allow fault tolerant message queuing and scalable N-N connections, as well as dynamic service discovery to provide full hot swapability for data rerouting and fault correction. The highly modular structure is fast and easy to develop for with supporting tools for remote control web interfaces and node management. The framework has seen successful deployment on the ANNIE experiment, where it has been used not just for DAQ but reconstruction, analysis, monitoring, post processing and event building. It has also been chosen as the DAQ software for Hyper-K and E61, where development is already mature. It is currently being used on multiple test stands and stand alone hardware components, with further interest from other experiments.

References

[1] B. Richards, *ToolDAQ Framework* [software] , **version 2.1.1**, (2018) https://doi.org/10.5281/zenodo.1482767

[2] ZeroMQ project, *ZMQ* [software], **version 4.0.7**, (2018). Available http://zeromq.org/intro:get-the-software [accessed 2018-07-09]

[3] K. Abe et al., *Hyper-Kamiokande Design Report*, (KEK preprint, February 2016) 2016-21

[4] NuPRISM and Hyper-K Collaborations, *An Intermediate Water Cherenkov Detector at J-PARC* ,JPS Conf.Proc. **12**, 010039 (2016)

[5] A. R. Back et al, *Accelerator Neutrino Neutron Interaction Experiment (ANNIE): Preliminary Results and Physics Phase Proposal*, (physics.ins-det, arXiv:1707.08222, 2017)