A Control Software for the ALICE High Level Trigger

Timm M. Steinbeck*, Volker Lindenstruth, Heinz Tilsner,
Kirchhoff Institute of Physics, Ruprecht-Karls-University Heidelberg, Germany,
for the ALICE Collaboration

Abstract

The ALICE High Level Trigger (HLT) cluster is foreseen to consist of 400 to 500 dual SMP PCs at the start-up of the experiment. The software running on these PCs will consist of components communicating via a defined interface, allowing flexible software configurations. During ALICE's operation the HLT has to be continuously active to avoid detector dead time. In order to ensure that the several hundred software components, distributed throughout the cluster, operate and interact properly, a control software was written that is presented here. It was designed to run distributed over the cluster and to support control program hierarchies. Distributed operation avoids central performance bottlenecks and single-points-of-failures. The last point is of particular importance, as each of the commodity type PCs in the HLT cluster cannot be relied upon to operate continously. Control hierarchies in turn are relevant for scalability over the required number of nodes. The software makes use of existing and widely used technologies: Configurations of programs to be controlled are saved in XML, while Python is used as a scripting language and to specify actions to execute. Interface libraries are used to access the controlled components, presenting a uniform interface to the control program. Using these mechanisms the control software remains generic and can be used for other purposes as well. It is being used for HLT data challenges in Heidelberg as well as during ALICE beamtests.

ALICE OVERVIEW

ALICE [1] [2] [3] is A Large Ion Collider Experiment for the future Large Hadron Collider (LHC) being built at CERN. It can be operated in both modes of LHC, proton-proton (pp) as well as heavy-ion (HI). The primary task of its High Level Trigger (HLT) [4] is the reduction of its input data stream of up to 25 GB/s to at most 1.2 GB/s which are accepted by the data acquisition for storage to tape. In order to achieve this goal three basic measures are taken by the trigger system: filtering of interesting events, selection of regions-of-interest in events, and online compression of the filtered and selected event data. For this purpose the HLT performs a full online event reconstruction from the raw data of the detectors participating in the trigger decision. A PC cluster of initially 400 to 500 nodes will be used to perform the analysis. The cluster nodes will be arranged in multiple hierarchy levels matching the detector layout as well as the sequence of analysis steps required. Processing of data will be performed by a processing chain made up of software components distributed over the cluster nodes [5] [6]. In order to manage the large number of software components active in the system the TaskManager control system described in this paper has been developed.

* timm.steinbeck@kip.uni-heidelberg.de

THE TASKMANAGER CONTROL SYSTEM

Requirements

For this control system a number of requirements have been defined which it should fulfill:

- Flexibility: The exact configuration in which the HLT will run has not yet been fixed. Furthermore it should be possible to create setups for small tests, data challenges or testbeams easily. In addition the system should retain the ability to be used for other projects as well.
- Hierarchy: The more than 2000 processes expected for the full HLT system will not be manageable in a sensible manner by a single supervisor instance. Additionally a good hierarchy can enable partitioning and significantly ease configuration of the system.
- Fault Tolerance: As the single PCs in a cluster are inherently unreliable the system should be designed from the start to avoid single-points-of-failure. This will allow the system to be built without single very reliable but also expensive components. It should also be possible to integrate TaskManager systems with other fault-tolerance or fabric-management software or hardware, e.g. the EU DataGrid's work package 4 [7] software or the system presented in [8].
- Finally, the system should be able to run alongside the analysis processes on the cluster nodes without impacting performance significantly.

Architecture

The architecture of the TaskManager system is outlined in Fig. 1. Around the core component three subsystems provide most of the functionality of the system: The Configuration Engine (CE), the Program State and Action Engine (PSAE), and the Program Interface Engine (PIE). XML [9] configuration files are read in by the Configuration Engine which provides the program's other modules with access to the file's configuration items. Contained

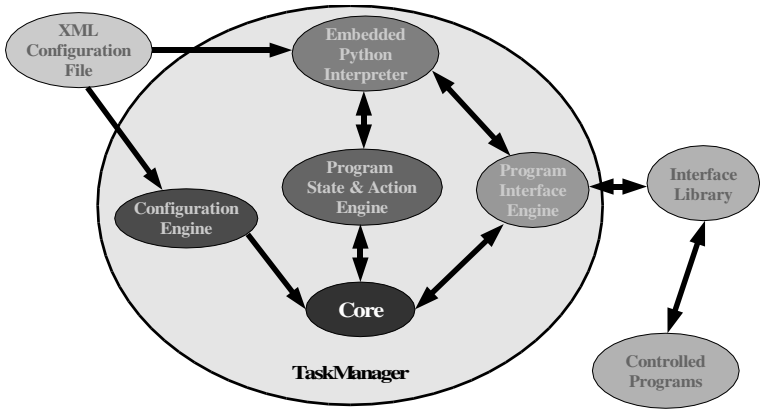

Figure 1: The Architecture of the TaskManager System

in the configuration file are sections of Python [10] code which are executed by an embedded Python interpreter. This execution is controlled by the Program State and Action Engine, which triggers it in response to certain events, e.g. state changes in programs. Communication with the controlled programs themselves is performed by the Program Interface Engine. It uses external interface libraries, supplied with the controlled programs, that offer a well-defined interface for communication.

The Configuration Engine

As already mentioned above, the TaskManager configuration is stored in XML files which are read in by the Configuration Engine. The other modules, including the core, can then query specific configuration items from the CE. Primarily a configuration file contains specifications of the programs to be started under the control of the system as well as specifications of Python code to be executed for specific events. Two more important items stored are the interface library used for communication with the programs as well as master-slave configuration items in hierarchical setups.

For each program that has to be run under the TaskManager's control a configuration entry holds several different sub-entries. The most important of these is the specification of the command that has to be executed for the program to run. Included in this are all command-line parameters required for the program's operation. For communication with the program its address has to be known to the system. It is handled only as an opaque string that is passed to the interface library which can thus interpret it as needed. The main part of the configuration will mostly be taken up by the different Python code sections. Several program events exist for which code can be specified. This code is executed when the given event occurs. Supported events include state changes, program termination, and changes in the program's configuration entry. Beyond these items, resources, e.g. files, used by the program can also be specified, so that they can be cleaned up after termination.

Next to the program entries and their respective sub-entries configuration files can also contain a number of global configuration entries. Primary entries in this area are the interface library used for program communication and a global Python section executed upon a state change in any program. For hierarchical setups the slave configuration also is one of the global configuration entries. The master configuration is not treated specially as it makes use of the normal mechanisms for program start and uses the interface library mechanism for master-slave communication.

The Program State and Action Engine

In many existing control systems classical state machines are used for modelling of programs states' and specification of actions to be taken on events, particularly program state changes. In the TaskManager system these tasks are performed instead by Python Actions, code specified in the configuration file. This Python code is executed by an embedded Python interpreter. Compared to a pure state machine this approach is much more flexible and powerful, supplying users with the full capabilities of the Python language, which is powerful as well as easy to learn and program in. Different events are supported in the system which will cause Python Actions to be executed. Among them are program state changes, either global or per program, program termination, or configuration changes. When such an event occurs the corresponding Action code is executed by the interpreter.

In order to allow access to TaskManager internal data, in particular to query programs' states and send commands to them, a set of special functions are exported by the TaskManager to the interpreter. These functions can be called by the Python code like ordinary Python functions. Python code can thus interact with the TaskManager system and/or controlled programs. Next to querying programs' states and sending commands a number of other functions are available. More detailed status data for a pro-

gram can be queried as well as its state. Programs can be started or terminated explicitly. Finally, data can be exchanged between master and slave TaskManager instances.

The Program Interface Engine

Different programs typically present different interfaces to outside programs for querying their state and status data and sending commands to them. This ranges from the standard Unix/Linux mechanisms of process state determination and signalling, over shared memory areas containing status information to background threads for handling TCP network connections for queries and commands. A mechanism has been developed for the TaskManager so that it can provide maximum flexibility and support for all kinds of programs. The mechanism uses interface libraries that are supplied together with the programs. As these libraries conform to a defined calling interface, supplying a set of specified functions, the TaskManager can use each of them in a similar manner. The libraries are implemented as shared libraries whose pathnames are specified in the configuration files. Using special operation system functions the TaskManager can load these libraries with their functions dynamically at runtime. After this loading process the functions contained in the library can be called as normal from C/C++ programs.

Among the interface library functions are the functions for querying program state and status data as well as sending commands to programs, which are called both from the TaskManager core as well as from the Python Action scripts. Beyond this functionality three major points of functionality are provided by library functions. Initialization and deinitialization of the library itself are typically called only once, at the start and termination of a specific TaskManager setup. The last major function group that has to be provided in interface libraries allows the TaskManager to wait for interrupt (or Look-At-Me (LAM)) messages from controlled programs. These messages will cause the TaskManager to query a program's state outside of its normal periodic polling schedule.

Master-Slave Operation

For master-slave operations two additional facilities are supplied by the TaskManager system, one each on the slave and master side. In slave TaskManager processes an additional component is active, the Master-Slave interface object. This component is responsible for receiving commands from master processes and dispatching them to the appropriate slave TaskManager subsystem. A second important functionality of this component is providing status data to master processes.

On the master process side no special extensions are needed inside the TaskManager process itself. Here the basic mechanism of communication with controlled processes via an interface library is used. A special interface library is provided that supplies the required support for master-slave TaskManager communication. Slave Taskmanagers are specified as any other process to be executed under the master TaskManager's control. By reusing the default mechanism the required configuration item support is reduced to a minimum on the master.

SAMPLE SETUP

An example of a TaskManager configuration is the setup that was intended to be used for the HLT activities during the beamtest of an ALICE TPC chamber prototype, outlined in more detail in [6]. In this setup two HLT PCs were used to obtain data from the detector via optical links from a DAQ PC. A third HLT PC received this data via TCP over Gigabit Ethernet, merged it, and transmitted it to another DAQ PC via identical optical links. On the three HLT PCs several software components are active, three on each of the two receivers and five on the sender. These components are controlled by a small hierarchical TaskManager configuration. In this configuration, shown in Fig. 2, one master TaskManager and three slave TaskManagers are used.

Each of the three slave TaskManagers supervises the local processes on one of the nodes. These TaskManager instances run directly on the same nodes as the processes they are supervising. Communication among slave TaskManagers and their processes is therefore purely local, although it is still handled via TCP/IP. But on Linux local TCP/IP communication bypasses some of the networking stack.

The master TaskManager process also communicates with its slaves over TCP/IP in this setup. It can therefore be run on any computer which has a network connection to the three HLT PCs involved. For simplicity it was also run on the third HLT PC, the merger. From here it communicates only with its slaves, it does not directly establish communication with the processes controlled by the slave TaskManagers.

As described in [6] the setup actually used for the HLT beamtest activity was different from this intended one, for due to the data size limit in DDL protocol. But the setup used functioned without any problems. As setups of similar or greater complexity are and have been used for local tests in Heidelberg as well, no problems are to be expected for the planned configuration as well.

SUMMARY AND OUTLOOK

The presented TaskManager control system provides flexible and hierarchical program control. It achieves this through two main features, the first of which is the use of an embedded Python interpreter for state and action handling. The second of these features is the use of interface libraries belonging to the controlled programs, making the TaskManager adaptable to most if not all programs. By using XML as the configuration file format both a quick and easy manual creation for small setup is possible as well as automated generation by programs for large configurations and/or autonomous operations. Additionally it can avoid single-points-of-failures in its configurations through

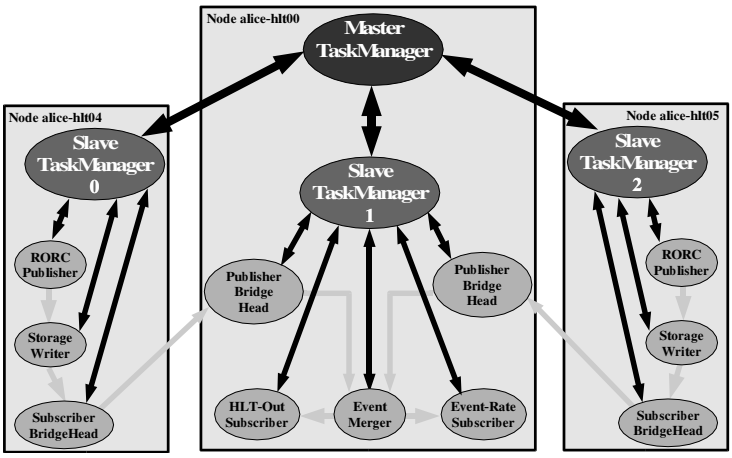

Figure 2: Configuration of the TaskManager system used during the TPC beamtest. Black arrows show the control flow of status information and commands, grey arrows show the flow of data.

its hierarchical design, increasing fault-tolerance and thus reliability. It has already been used for ALICE HLT data challenges as well as for the HTL activity during a TPC prototype testbeam. But it has also been designed to be adaptable to a wide range of problems, inside the ALICE HLT as well as for unrelated projects outside of it. The TaskManager package together with auxiliary packages is available from [11], documentation can be found at [12].

ACKNOWLEDGEMENTS

Work on the ALICE High Level Trigger has been financed by the German Federal Ministry of Education and Research (BMBF) as part of its program “Förderschwerpunkt Hadronen- und Kernphysik - Großgeräte der physikalischen Grundlagenforschung”.

REFERENCES

[1] http://ALICE.web.cern.ch/ALICE/.

[2] http://ALICE.web.cern.ch/ALICE/user.html.

[3] The ALICE Collaboration, “ALICE - Technical Proposal for A Large Ion Collider Experiment at the CERN LHC”, CERN/LHCC/95-71, LHCC/P3, December 1995.

[4] The ALICE Collaboration, “ALICE - Technical Design Report of the Trigger, Data Acquisition, High-Level Trigger, and Control System”, CERN/LHCC/2003-062, January 2004.

[5] T. M. Steinbeck, V. Lindenstruth, H. Tilsner, “A Software Data Transport Framework for Trigger Applications on Clusters”, CHEP03, La Jolla, USA, 2003.

[6] T. M. Steinbeck, V. Lindenstruth, H. Tilsner, “New experiences with the ALICE High Level Trigger Data Transport Framework”, CHEP04, Interlaken, Switzerland, 2004.

[7] http://hep-proj-grid-fabric.web.cern.ch/hep-proj-grid-fabric/.

[8] R. Panse et al, “A Hardware Based Cluster Control And Management System”, CHEP04, Interlaken, Switzerland, 2004.

[9] http://www.w3.org/XML/

[10] http://www.python.org/.

[11] http://www.ti.uni-hd.de/HLT/software/software.html

[12] http://www.ti.uni-hd.de/HLT/documentation/documentation.html