PROCEEDINGS OF SCIENCE

Monitoring the software quality in FairRoot

Florian Uhlig*
Gesellschaft für Schwerionenforschung, Plankstrasse 1, 64291 Darmstadt, Germany
E-mail: f.uhlig@gsi.de

Mohammad Al-Turany
Gesellschaft für Schwerionenforschung, Plankstrasse 1, 64291 Darmstadt, Germany
E-mail: m.al-turany@gsi.de

One of the challenges of software development for large experiments is to manage the contributions from globally distributed teams. In order to keep the teams synchronized, a strong and immediate quality control is important to find problems when they are introduced in the code base. This quality control includes checks if the project can be build from the sources but also if the executable can be executed and delivers the expected results. The requirement to do these tests frequently and on many different platforms immediately results in the necessity to do these checks automatically.
With an increasing number of supported platforms it becomes impractical to maintain installations of all these platforms for quality assurance. The easiest way to overcome this problem is to use a client server architecture. The clients for each platform to be tested do the build and test step and send the collected data in the end to a central server. This central server is responsible for the storage, the processing and in the end for the presentation of the data.
In this article, we present the scheme which is used within the FairRoot framework to continuously monitor the status of the project. The tools used for this task are based on the open source tools CMake and CDash.

13th International Workshop on Advanced Computing and Analysis Techniques in Physics Research
February 22-27, 2010
Jaipur, India

*Speaker.

PoS(ACAT2010)043

1. Introduction

The proposed project FAIR (Facility for Anti-proton and Ion Research) is an international accelerator facility of the next generation [1]. It builds on the experience and technological developments already made at the existing GSI facility, and incorporates new technological concepts. The four scientific pillars of FAIR are NUSTAR (nuclear structure and astrophysics) [2], PANDA (QCD studies with cooled beams of anti-protons) [3], CBM (physics of hadronic matter at highest baryon densities) [4] and APPA (atomic physics, plasma physics, and applications) [5].

The FairRoot framework [6,7,8] is an object-oriented simulation and analysis framework for experiments planned to be built at FAIR. It is based on ROOT [9], and the Virtual Monte-Carlo (VMC) [10] interface. It includes core services for detector simulations and offline analysis. The framework, is designed to optimize the accessibility for beginning users and developers, to increase the flexibility, to cope with future developments, and to enhance synergy between the different physics experiments at or outside the FAIR project.

Beside the framework itself the FairRoot core team also provides the tools to build and test the experiment specific software base on FairRoot. The build system is based on the open source software CMake [11,12]. CMake itself includes the capability to automatically compile and test the project. This creates the possibility to implement an automatic quality assurance of the software on many different platforms.

Platform in this sense means a combination of operating system, operating system flavor, compiler, and compiler version. The supported operating systems are Linux in nearly any flavor and MacOS X 10.5 and 10.6. There is also an experimental port to Open Solaris. There are no known restrictions concerning the compiler. The code was already tested with the GNU compiler collection (gcc), the Intel compiler collection (icc), and the Sun Studio compiler suite, however the mainly used compiler is from gcc.

The collected results from any of these different platforms are sent to a central web server. This web server uses the php base open source tool CDash [13] and a database to store, to process and display all the collected information.

The complete build and test system is based on the open source tools CMake, and CDash.

2. Build System

FairRoot started as many projects with Unix makefiles, but after some time it became obvious that it would need too much manpower to maintain these makefiles for several different platforms. We evaluated several programs to generate makefiles in an automatic way depending on the platform. Finally we chose CMake as build system for FairRoot since it provides all needed features, it is open source and available for all our target platforms.

CMake is an open-source system that manages the build process in an operating system and in a compiler independent manner. Simple configuration files in a portable text format are used to generate native build files. These can be either project files for Integrated Development Environments such as Eclipse, Apple XCode or Microsoft Visual Studio, as well as UNIX, Linux or NMAKE style makefiles. Due to it's modular design it is easily extensible. CMake comes with an integrated scripting language which has the ability to create complex custom commands for auto-

matically generated files such as ROOT dictionaries. These commands can be used to generate new source files during the build process that are then compiled into the software. CMake also includes the functionality to execute other external programs. This is for example used to automatically generate the class documentation of FairRoot on a daily basis using Doxygen [14].

3. Automatic Tests

Since FairRoot is used by all of the big FAIR experiments as base for there own specific developments it is essential to know the status of the project at any time to find problems which enter the code base as early as possible. This includes especially changes in the external dependencies and problems which occur only on some of the supported platforms. To produce this quality assurance (QA) information as the first step we actually use the tool CTest which is part of the CMake suite. The second step is to collect all the created information, store them, and provide the information for later usage.

Since the QA information should be created at any time they are useful, there are three different build modes which are supported. The first one is the so called *Experimental Build* for developers who experiments with new features and want to build and test the code automatically. They can trigger the *Experimental Build* on there machine which then will automatically configure (create Makefiles using CMake), compile and test the code, which can be different from the code in the repository, on their system. To test the project the same ROOT macros are used as for the normal simulation and reconstruction. The test stage is easily extensible by adding new ROOT macros or external programs.

Whenever there is a change in the central software repository, the repository triggers a *Continuous Build* on a dedicated build server which automatically updates the local working copy from the central software repository, and after that configures, compiles, and tests the project. To decrease the time needed for the compile step the *Continuous Build* does not start with a clean build directory but reuses the build directory from the last run so that only files which have changed have to be compiled again.

Once per day (normally during night when the machines are idle) we run a *Nightly Build* on all available platforms. In contrast to the *Experimental Build* and *Continuous Build* we start always with a clean build directory. Also the tests performed can be different between the three test modes.

After any of the three test modes the generated information about the configure, compile and test stage are sent to the central web server for further usage. The information contains the output from the generation of the Makefiles, all warnings and errors from the compilation stage, and the results from the test stage. The build stage will not fail completely if there is an error but will go on with the next target which is not affected by the error. In the end one will get all error messages which occur during the build stage. After the build stage one may have a project with missing libraries or executables. This can then affect some of the tests but others will run without any problems. The results from the test stage include information if the test passed, failed or ran into a timeout together with the runtime for the test and in case of a failure or a timeout the complete output of the test.

The advantage of this client server architecture is, that each user can setup his own test system. This makes it possible to test many different platforms without the necessity to have direct access

to the system. This also makes sure that the environment on the user system is exactly the same as the one on which the test is done. If both systems are different there is always the possibility that there are slight differences between the two systems which results in endless debug sessions to find these differences. So using this client server architecture each user can help to improve the project when setting up a test system for an up to now untested platform. This setup is done adjusting one template according to the setup of the specific machine and creating a cron job for automatic execution of the build and test script.

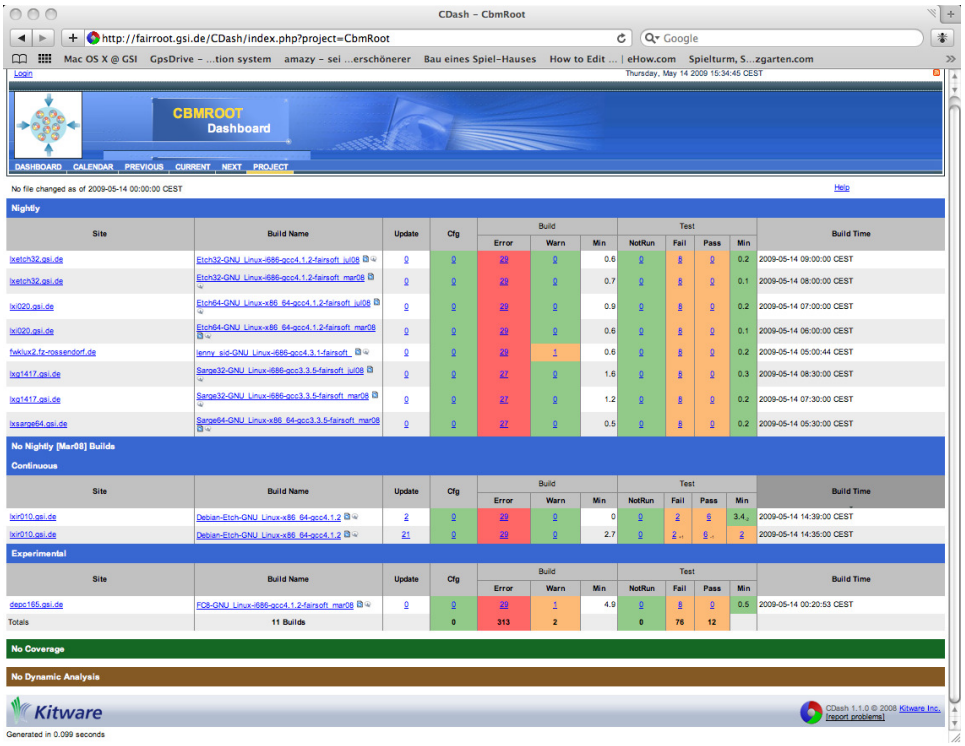

Figure 1: Overview page of the FairRoot Dashboard.

4. Dashboard

The second and not less important part of the quality control is to collect all the produced information, to process them, and as final result to display them in an easy to use fashion. For these tasks the software package CDash is used together with a database to store the collected information. CDash on request generates dynamic web pages out of the stored information. As an example the main page of the so called Dashboard is shown in figure 1 for a given date. This main page provides a fast overview about the status of the project. From here one can get more detailed information e.g. about updates or problems during the configure, build and test stage.

From the generated information sent by the clients the server can produce additional aggregated information as for example the runtime for a specific test on a specific machine as a function of time as shown in figure 2. In the figure one can clearly see two hot spots were some changes

in the code result in a much longer run time of the test. A problem which can sneak in the code unnoticed if no automatic testing is done. All the available information enables the developers to spot problems in a fast and convenient way. The client server architecture of the QA process enables developers to test the code on all available platforms, even if one has no direct access to the specific platform.

The server can also produce automatic alerts to administrators or developers in case of problems. This can be e-mails to a developer who commits erroneous code or e-mails to site maintainers if a machine expected to submit results did not do so.

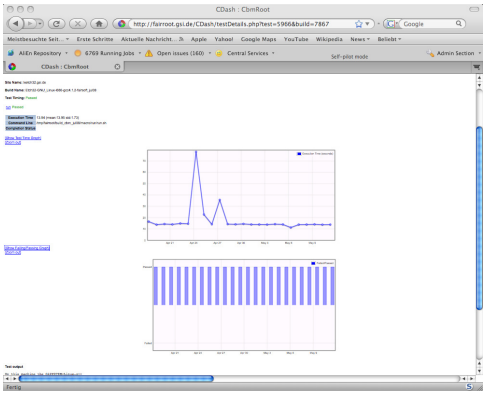

Figure 2: Detailed run time information for a specific test.

The idea with the Dashboard is to have only one point where all available information is accessible. This also includes direct links from the generated web pages to the web front end for the SVN repository. This allows to go from an error or a warning directly to the problematic file. The Dashboard also provides direct access to all kind of other information like the class documentation generated with Doxygen or the results about the coding convention compliance check.

5. Summary

In summary the build and test system of FairRoot was described which is completely based on the open source software CMake. One of the big advantages of CMake is the huge flexibility. It easily allows to include the production of the Doxygen documentation or the check of the coding conventions into the FairRoot build framework using only the built in macro language of CMake. The build system provided by FairRoot is also used by the experiments which uses FairRoot as base of their own experiment specific developments. This enables these experiments to use all the functionality like the test support or the dashboard out of the box.

In table 1 we summarize information about two experiments using the FairRoot build system and compare them with results from the ALICE experiment [15] which uses also a CMake based build system. Shown in the table are the number of targets which have to build for each of the experiments. A target is either a shared library or an executable. Together with this information also the time to configure the project and the compile times are give for a normal and a parallel build. To get an idea about the size of the different projects the number of lines of code (LOC) is

Experiment	Targets	configuration time	build time -j1	build time -j8	LOC
CBM	29	2s	612s	92s	$\approx$ 270.00
Panda	61	4s	730s	97s	$\approx$ 370.00
ALICE	203	20s	2690s	340s	$\approx$ 2.500.00

Table 1: Configure and compile times for different experiments using CMake.

given as reference. This number was achieved using the tool cloc [16] and is the number of lines of code excluding comment and blank lines. All tests concerning the configuration and compile times have been done on a 8-core Intel Xeon with 2.67 GHz.

Depending on the different build machines submitting to the Dashboard the build times range from 1.5 to 20 minutes for CBM and 1.5 to 10 minutes for Panda. The number of tests depend very much on the experiment and is normally increasing with time. Up to now no unit tests are implemented as all tests are system tests. The time per test varies between some seconds and approximately 5 minutes which results in a large time spread for the complete test times (4 to 45 minutes for CBM and 10 to 60 minutes for Panda). The main reason for this large spread is that the tests cannot run in parallel. This limitation should disappear with the newest version of CMake but is not tested for the FairRoot build system. The approximate number of tested platforms for CBM and Panda is around 20. This number is always changing because new platforms are used by developers and old ones are switched of. Since the test platforms are not maintained by the FairRoot core team but by the different users always the platforms of interest are tested.

References

[1] FAIR: http://www.fair-center.de/

[2] NUSTAR: http://www.gsi.de/forschung/fair_experiments/NUSTAR/index_e.html

[3] PANDA: http://www-panda.gsi.de/

[4] CBM: http://www.gsi.de/forschung/fair_experiments/CBM/index_e.html

[5] APPA: http://www.fair-center.eu/APPA-Physics.187.0.html

[6] M. Al-Turany, D. Bertini, and I. Koenig. CbmRoot: Simulation and analysis framework for CBM experiment. In S. Banerjee, editor, *Computing in High Energy and Nuclear Physics (CHEP-2006)*, volume 1 of *MACMILLAN Advanced Research Series*, pages 170-171. MACMILLAN India, 2006.

[7] D. Bertini, M. A-Turany, I. Koenig, and F. Uhlig. The fair simulation and analysis framework. In *International Conference on Computing in High Energy and Nuclear Physics (CHEP'07)*, volume 119 of *Conference Series*. IOP Publishing, 2008.

[8] M. Al.-Turany FairRoot: http://fairroot.gsi.de.

[9] R. Brun and F. Rademakers. Root - an object oriented data analysis framework. *Nuclear Instruments and Methods in Physics Research A*, 389:81-86, Sep. 1997.

[10] R. Brun, F. Carminati, I. Hrivnacova, and A. Morsch. Virtual Monte-Carlo. In *Computing in High Energy and Nuclear Physics*, pages 24-28, La Jolla, California, 2003.

[11] CMake: http://www.cmake.org

[12] K. Martin and B. Hoffmann. Mastering CMake, A Cross-Platform Build System, Kitware Inc., 2007

[13] CDash: http://www.cdash.org

[14] Doxygen http://www.doxygen.org

[15] ALICE: http://aliceinfo.cern.ch/Collaboration/index.html

[16] cloc: http://cloc.sourceforge.net/