Experiments Toward a Modern Analysis Environment: Using TMVA and other tools in a functional world with continuous integration for analysis

Gordon Watts
University of Washington/Seattle

Abstract

A modern high energy physics analysis code is complex. As it has for decades, it must handle high speed data I/O, corrections to physics objects applied at the last minute, and multi-pass scans to calculate corrections. An analysis has to accommodate multi-100 GB dataset sizes, multi-variate signal/background separation techniques, larger collaborative teams, and reproducibility and data preservation requirements. The result is often a series of scripts and separate programs stitched together by hand or automated by small driver programs scattered around an analysis team's working directory and disks. Worse, the code is often much harder to read and understand because most of it is dealing with these requirements, not with the physics. This paper describes a framework that is built around the functional and declarative features of the C# language and its Language Integrated Query (LINQ) extensions to declare an analysis. The framework uses language tools to convert the analysis into C++ and runs ROOT or PROOF as a backend to determine the results. This gives the analyzer the full power of an object-oriented programming language to put together the analysis and at the same time the speed of C++ for the analysis loop. A fluent interface has been created for TMVA to fit into this framework, and can be used as a model for incorporating other complex long-running processes into similar frameworks. A by-product of the design is the ability to cache results between runs, dramatically reducing the cost of adding one-more-plot. This lends the analysis to running on a continuous integration server after every check-in (Jenkins). To aid to data preservation a backend that accesses GRID datasets by name and transforms has been added as well. This paper will describe this framework in general terms along with the significant improvements described above.

Physicist

- Checks in new analysis code after running locally.
- This is the primary way to interact with the analysis system.
- Either a job is fired off automatically by the check in, or physicist does it manually when they are ready.
- Only results from the CI server are used: thus no hand-art (for the paper, CONF, talks…)

Git/SVN/etc.

- Stores all source code for all jobs
- Can store scripts to run jobs

Build Job

- Run in a Virtual Machine: isolated, controlled environment.
- Assures results are repeatable and there is no hand art.
- As input can use files from other build jobs.
- Or large ROOT-tuples in a known location on large servers.
- Jobs can run on Linux or Windows

User Programs

- Access from any internet connected computer.
- Use the REST API to fetch a particular ROOT file or other output file.
- Generate plots, run tests, etc.

Jenkins Build Server

A build server or continuous integration server (CI) is a **batch job manager/server**.

- You give it a set of instructions to run a job.
- You tell it where in source control to find all the source code for the job.
- You tell it what files are important to archive and save at the end of a successful job.
- It will run the job, perhaps with some parameters you specify.
- It will save the log file, version information from source code used, and important output files.
- A job can be triggered by a web API or automatically by a check-in to source control.
- Jenkins, while powerful, is missing enterprise grade security and complex job workflow.

REST API

- Any external program can trigger build, retrieve log files and result files
- Authenticated with standard web security.
- Jobs run on the build server can fetch data files this way.

Jenkins Build Data

Job Info

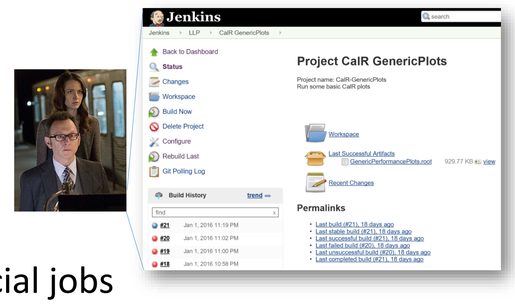
- Parameters used to run
- Commands Executed
- Log file of job
- Source Control Tags of everything run
- Date, time taken, etc.

Important Files

- All files from all jobs kept
- Easily accessible over the web with username

Web Interface

Uses the standard Jenkins web interface. Available anywhere in the world with a log-in. Shared with other analysis team members.



- Fire off special jobs
- Input custom parameters
- Configure jobs (change commands, etc.)
- Look at previous jobs config
- Easy access to log files
- Each access to archived files from each job
- Maintenance (prune saved job directory, etc.)

Data Preservation Features

Official ATLAS Production Dataset
(has unique name)

Analysis TTree Maker

- Runs GRID job from code stored in ATLAS SVN
- Resulting dataset gets unique name
- Software does query to ATLAS job system (BigPanda) to figure out what has been run and hasn't.
- Big problem: user datasets sometimes remain on GRID for less than a month.
- Command line tools (on Windows!) compiles and submits GRID job with just ATLAS dataset name, job name, and job version.

```
job(DiVertAnalysis,8)
{
    release (Base,2.3.53)
    package(atlasphys-exo/Physics/Exotic/UEH/DisplacedJets/Run2/AnalysisCode/trunk/DiVertAnalysis,
        263193)

    submit(DiVertAnalysisRunner -EventLoopDriver GRID *INPUTDS*
        -ELGRIDOutputSampleName *OUTPUTDS* -WaitTillDone FALSE -isLLPMC true)
}
```

Jenkins Analysis Job

- Uses the official ATLAS dataset name, the TTree job name, and the TTree job version
- Job fails if the dataset hasn't been produced on the GRID (or has been deleted from the GRID).
- Jenkins job now contains the input dataset, GRID jobs run on them, and the output ROOT file of produced plots, **enabling end-to-end tracking**.

Status & Future

- A recent analysis was released by ATLAS using these tools (ATLAS-CONF-2016-103).
- All the MVA training and limit setting and lifetime extrapolation was done with these tools.
- **Disaster struck**: Two days after the release of the CONF note a power outage occurred at UW. Despite a UPS protecting it, the disk where the build machine VM was stored did not recover.
- **Future**: Investigate cloud options for running the build server.
- Complex workflow: Jenkins doesn't support it well.
- **Conclusion:** This system has many advantages… How to bring them to the **mainstream**?

Building After Every Check-in

- It is very useful to build after every checking, just like on a real build server.
- You can track exactly what each source code change does.
- **Major problem**: what if your job takes 5 hours to re-run?
- **Insight**: A job consists of 1000's of plots. Most source code changes affect 10 plots.
- Re-run to update the 10 plots, and copy over the other unchanged plots.
- Much faster. But requires a very different programming model.
- LINQToROOT enables this (including fluent TMVA integration).