DISTRIBUTING SOFTWARE APPLICATIONS BASED ON RUNTIME ENVIRONMENT

N. Ratnikova *, R. Darwish, D. Evans, FNAL
N. Darmenov, CERN, T. Wildish, Princeton

Abstract

Packaging and distribution of experiment-specific software becomes a complicated task when the number of versions and external dependencies increases. In order to run a single application, it is often enough to create appropriate runtime environment that ensures availability of required shared objects and data files. The idea of distributing software applications based on runtime environment is employed by Distribution After Release (DAR) tool. DAR allows to automatically replicate application's runtime environment based on the reference software installation. Assuming that software is relocatable, applications can be packaged into a completely self-consistent "DAR-ball" and executed on any system, which is binary compatible with the reference software installation. Such lightweight distributions can be used on opportunistic GRID resources to avoid excessive efforts of complete installation of experiment-specific software. For over three years, DAR tool has been successfully used by CMS for Monte-Carlo mass production, helping physicists to get results faster. In version 2, DAR was completely redesigned, optimized, and enriched by the new features, ready to meet future challenges. The paper presents general concept of the tool and new features available in DAR-2.

INTRODUCTION

State of the art

Software development and deployment in a large scientific collaboration has become more complex and more challenging task in the era of distributed computing. The High Energy Physics experiment specific software is characterized by a large number of interdependent software components and rapid development cycle, especially in the early phases of the experiment. Data intensive applications require enormous CPU resources for the computational tasks, and sophisticated storage systems for storing and maintaining the data, and making them available to users.

One of the ways to cope with the large-scale computational problems is to utilize available GRID [2] resources and technologies. In the GRID environment computational tasks are represented as jobs that can be sent to various computing sites depending on available CPU resources and/or required data.

It is the goal of the software distribution process to provide software availability on the remote execution sites.

Regardless of the implementation techniques, the software distribution process includes the following steps:

- Packaging
- Transfer
- Installation
- Publishing

The packaging step usually includes creation of the archive to be transferred to the execution site, and the installation instructions. The installation step includes unpacking, local configuration, and some verification procedure.

It is often convenient to have verification as a separate step, as it could then be repeated later as needed without re-installing the software.

Concept

An important requirement of distributed computing is that application software must be relocatable.

Most common way to handle relocatable software is to set the environment variable that tells the application where to look for required files. Examples are standard UNIX variables such as `LD_LIBRARY_PATH` variable that tells the loader where to look for shared libraries, `PYTHONPATH` that defines search path for Python modules, `PATH` that defines search path for executables. Many other variables are being used to specify the locations of required data, configuration files, plug-ins, and so forth.

In CMS [1] the entire runtime environment for the software applications is generated by SCRAM, Software Configuration, Release, And Management tool [3]. The software configuration assures that the runtime environment is set in a way that all required files can be found at the execution time.

In a nutshell the idea of distributing software applications based on the runtime environment is the following: all files associated with the application runtime environment are being packaged, and then deployed in a new location. The same environment variables are then set to point to the new locations. A well-designed and truly relocatable application will run in a new environment exactly as if in the original one.

Similarly, the application runtime environment can be deployed on another node with the same, or compatible operating system. The operating system compatibility here means that the same application will run on both systems, and produce the same result.

* natasha@fnal.gov

Described approach allows to package and deploy all necessary files and reproduce exactly the same runtime environment for the software application on the target site. Essential point here is that the object of the distribution is a self-contained application. This is different from the conventional generic approach of reproducing the complete software development, when multiple software components are distributed in a modular way [4].

There is no need to compile any parts of code on the remote site. The public applications and user-specific application are handled in the same way.

There is no need to deal with the internal dependencies between the products used in the application: all these issues are being already resolved by the time when the runtime environment is defined.

The runtime environment based distributions are usually more compact, since they to not need to include the whole development environment, and the variety of versions.

The abstraction from development environment details makes the runtime environment based distributions very robust.

TOOL DESCRIPTION

DAR is a software tool that implements the concept of distributing applications based on runtime environment. Using a simple configuration file and command line options user can automatically package and install software applications on the execution host. To gather information about the runtime environment of the application DAR uses SCRAM. Based on this information DAR generates a DARball file, that can be used to recreate the same runtime environment on the machine where the application needs to be executed. This allows developers and users to trust that their application sent for execution on the GRID, will run identically on remote machines, and exhibit the same behavior as it does when run locally.

Packaging

Packaging of the application is done in one command. User provides the specification file with the runtime environment variables settings plus additional optional directives (see Fine Tuning). For applications developed within SCRAM managed projects it is sufficient to provide the project top development area as an argument. DAR will automatically generate the specification file using built-in `SCRAM2DAR` interface, and proceed with packaging. DAR will analyze the request contained in the specification file, create a list of file system objects associated with the values of the environment variables, taking into account additional directives. The resulting files are included into archive along with additional information saved as meta data Upon successful completion DAR informs the user about the locations of the DARball, respective log files, and installation instructions.

Installation

Installation of a DARball is done in one command. User must specify two arguments: the DARball, and the installation directory. DAR will unpack the distribution and create the runtime environment setup scripts. Upon successful completion DAR will print out instructions on how to set the application runtime environment.

The DARball

The DARball is essentially a compressed archive of files. It contains files associated with the runtime environment: shared libraries, executables, and other data files, and distribution meta data.

The internal structure of distribution represents both the runtime environment and the file system with preserved relative locations of the directories and files as in the original reference installation on the packaging site.

The meta data are stored in a separate directory. They include:

- application information, such as project name, release name, application name, etc .
- configuration information, such as versions of used external tools for SCRAM managed projects, architecture identifier.
- specification file used for DARball creation
- packaging details: starting time, creation time, host name, user name, DAR version used for packaging, log files
- installation size
- bill of materials with md5sum checklist
- templates for runtime environment setup scripts

All the contents are stored under the architecture specific directory.

Incremental DARballs

This is a new option implemented in DAR2 provided to create an incremental distribution based on existing DARball. The resulting incremental DARball is organized in the same way as described above, and contains complete information about the application runtime environment, but the files that are common with the base DARball are replaced by a reference to the corresponding files in the installation base. Two files are considered to be common if they have identical relative path, name, and md5sum. All references in the incremental DARballs are pre-generated in advance. This is done to insure the robust behavior during the installation.

The installation of the incremental DARball never modifies the base installation. It only re-uses already installed structure.

The md5sum check of the incremental installation can be done at any time to insure that all references are still valid.

The use of incremental DAR installations is rational when two applications have many common files. For example user specific application usually heavily re-use the public code. Or for executing slightly modified applications, when most of required files stay unchanged.

Incremental installation help to save disk space, while providing the same functionality as the traditional DARball installations.

Fine Tuning

By default DAR only uses the runtime environment information to create the distribution. In some cases user may know nuances about his application, that allow to reduce the distribution size. DAR provides a number of smart options to control the contents of the distribution.

- exclude files or directories matching specified patterns, for example : lib*.a , *.pyc, *.tgz, *.PDF, CVS
- preserve files matching a specific pattern. These files will not be replaced by reference, or excluded. They will be included.
- ignore all files referred to via a given environment variable: these files will not be included into the distribution, unless they are referred to by another environment variable.

Extra Features

Additional DAR features include

- built-it help information system.
- verbosity control.
- test mode for packaging step to quickly validate the new environment without copying all the files
- print out application information contained in DARball without doing the actual installation
- API (in Python) for interoperability or re-use within other tools
- produces the log files, including the time log for tracking the performance.
- support for multiple architectures
- platform independence

USE

DARballs are light-weight distributions and can be used for quick and easy software deployment anywhere where the software does not need to be rebuilt. They are especially useful on opportunistic GRID resources, as they allow to avoid excessive efforts to install complete software development environment.

DAR has been successfully used for four years in CMS for worldwide distributed Monte Carlo data production [5], [6].

Other uses include running demo applications, running analysis jobs in batch mode, benchmarking [7], and platform compatibility tests.

SUMMARY

The concept of distributing software based on runtime environment proved to work well for extended period of time, and has been adopted for use in various projects [5], [6], [7], [8], [9] .

DAR tool that implements this concept is very robust and easy in use. New version DAR2 re-implemented in Python provided new features, better performance and improved interoperability with other tools.

Incremental distributions, and smart options facilitate optimal use of the resources. Object-oriented design and provided API allow for easy extensions and re-use .

ACKNOWLEDGEMENTS

We would like to thank the CMS Production team, CMS Grid sites, and all other DAR users for valuable feedback.

REFERENCES

[1] Compact Muon Solenoid Collaboration http://cmsdoc.cern.ch/cms/outreach/html

[2] LHC Computing GRID http://lcg.web.cern.ch/lcg/

[3] SCRAM http://cmsdoc.cern.ch/Releases/SCRAM/doc/scramhomepage.html

[4] A. Nowack, et al. CMS Software Packaging and Distribution Tools.

[5] CMS Production http://cmsdoc.cern.ch/cms/production/www/html/general

[6] N.Ratnikova, A. Afaq, G. Graham, T. Wildish, V. Lefebure. Software Packaging with DAR. NIMA 534:110-114,2004

[7] H. Wenzel, M. Furukawa. Benchmarking AMD64 and EMT64, CHEP06 http://home.fnal.gov/ wenzel/Benchmarking

[8] N. Ratnikova, A.Sciaba, S.Wynhoff: Distributing Applications in Distributed Environment. NIMA 502: 458-460, 2003

[9] N. Ratnikova, G.Graham: CMS Software Distribution and Installation Systems:Concepts, Practical Solutions and Experience at Fermilab as a CMS Tier 1 Center. CHEP01, Beijing, Sept, 2001.