ATLAS software configuration and build tool optimisation

Grigory Rybkin
(on behalf of the ATLAS Collaborartion)

Laboratoire de l'Accélérateur Linéaire, Université Paris-Sud
CNRS/IN2P3, Orsay, France

Introduction

- ATLAS experiment [1] software code is over 6 million lines mainly in C++, Fortran, and Python and organised in over 2000 packages
- ATLAS uses Configuration Management Tool (CMT) [2] to configure and build its software
- CMT build performance is crucial because of software size and since ATLAS maintains or develops 4–5 full software release branches and $\geq$ 5 smaller physics analysis releases carrying out their integration builds for several platforms (with different compiler versions, optimised, for debugging) on a daily basis [3]
- performance of CMT setup and querying commands is very important and may even be critical since they are used when launching every data production or analysis software job, in particular, on Worldwide LHC Computing Grid (WLCG) sites, by various other tools, e. g., [4, 5, 6], as well as by the software users and developers interactively
- we describe work on CMT optimisation and present results achieved

Build time optimisation

How CMT works

- invokes CMT commands to generate Makefiles—according to requirements files—that, in turn, may contain CMT command invocations
- passes Makefiles to Make [7] to do package build—create libraries, applications, and perform actions (commands)
- such CMT command invocations may be quite numerous, e. g., for AtlasEvent project with 358 packages number is 6320 . Average number of CMT command invocations being about 18 per package
- originally, upon *each* CMT command invocation, CMT read requirements files of all used packages

New package build procedure

- package build procedure re-designed to introduce CMT command that reads effective configuration and generates *cached* requirements files that do not reference any other requirements files and contain all necessary information for subsequent CMT build command invocations
- this reduced the number of requirements files reads to (as a rule) *one* per used package
- in context of ATLAS software, on average, build command reading package requirements files spends $\approx$ 1.0s, while reading *cached* requirements file it spends $\approx$ 0.1s, i. e. $\sim$ 10 times less
- also, by combining some of the previously used commands in new commands, number of CMT build command invocations was reduced
- as a result, CMT build time overhead decreased $\approx$ 2.3–3.0 times

More build parallelism

- originally, the way to build a CMT project was to run "cmt broadcast make" in the package with dependencies on all the project packages. This command builds one package at a time, taking into account packages dependencies
- introduced new so-called BCAST build mode in which build is run with simple "(cmt) make", is performed on several independent packages at same time
- depending on project structure and number of processors available, this mode may reduce full build time by several times compared to usual "cmt broadcast make"

Even more build parallelism

- further modified package build procedure so as to compile even dependent applications and libraries in parallel whenever possible
- dependencies are imposed at link stage only
- this allows for more fine-grained parallelism at package task level

Default setup

- now CMT runs Make with "-j NPROCESSORS" option launching NPROCESSORS build processes in parallel
- so-called QUICK mode is enabled meaning that all Makefiles are only regenerated if any requirements file changes. Source files dependencies are still recalculated if a file or any dependency changes. This ensures faster and more efficient development cycle

CMT commands optimisation

- using program profiling tools [8], identified most time-consuming parts
- common for all the commands was function extensively used in requirements parsing files re-written making use of standard library function
- another improvement that affected virtually all commands was to introduce cache in order to optimise use of function accessing file system
- algorithm of heavily used command that generates source files dependencies revisited, in particular, replacing use of one standard library function with another standard library function performing considerably better

Results are the following:

- running setup is $\approx$ 1.7 times faster, number of *stat*, *lstat* calls reduced by $\approx$ 2.5 times—addresses problem of long setup time at WLCG sites, in particular
- querying commands like "cmt show ..." are $\approx$ 2 times faster
- "cmt build dependencies" command is $\approx$ 4 times faster

In addition, enhanced mechanism of source files dependencies generation at compilation time was implemented, giving full build time gain $\approx$ 10% (for C/C++ based projects)

- on average, time gain is $\approx$ 1 hour for each ATLAS full release branch

Runtime environment setup

- in order to cache environment setup, *-requirements* option was implemented so that "cmt -requirements setup" command generates standalone, or cached, requirements file that can then be used with "cmt setup" command for environment setup script generation
- especially useful for deployment on distributed file systems like AFS or CERN Virtual Machine File System (CERN VMFS) [9]

Conclusions

- significant build performance optimisation achieved thanks to CMT build procedure re-design based on making use of caching and parallelism
- CMT overhead decreased by $\approx$ 5 times and represents less than 1–5 % of full build time
- enhanced build parallelism at package task and package levels allows CMT to use the multi-core machines resources efficiently and reduces full build time by several times
- thanks to code optimisation, most of the CMT commands, in particular, configuration querying commands and runtime environment setup became $\approx$ 2 times faster. The latter can also be cached allowing for further optimisation
- these optimisations allow ATLAS to successfully build, develop, and use its software, and considerably improve software developer and user experience

References

1. ATLAS experiment http://cern.ch/atlas
2. Arnault C 2001 Experiencing CMT in software production of large and complex projects *Proc. Int. Conf. on Computing in High Energy and Nuclear Physics CHEP 2001* (Bejing, China), http://www.cmtsite.net/
3. Luehring F, Obreshkov E, Quarrie D, Rybkine G and Undrus A 2010 Organization and management of ATLAS nightly builds *J. Phys.: Conf. Series* **219** 042045
4. Albrand S et al. 2010 Organization, management, and documentation of ATLAS offline software releases *J. Phys.: Conf. Series* **219** 042012
5. Arnault C, De Salvo A, George S and Rybkine G 2004 The deployment mechanisms for the ATLAS software *Proc. Int. Conf. on Computing in High Energy and Nuclear Physics CHEP 2004* (Interlaken, Switzerland)
6. Rybkin G 2012 ATLAS software packaging *J. Phys.: Conf. Series* **396** 052063
7. GNU Make http://www.gnu.org/software/make/
8. GNU Gprof from GNU Binutils http://www.gnu.org/software/binutils/, Callgrind http://www.valgrind.org/
9. De Salvo A et al. 2012 Software installation and condition data distribution via CernVM FileSystem in ATLAS *J. Phys.: Conf. Series* **396** 032030