A continuous integration system for MPD Root: experiments with setup configuration

A. Degtyarev, G. Fedoseev, O. Iakushkin[a], V. Korkhov

Saint Petersburg State University, 7/9, Universitetskaya emb., Saint Petersburg, 199034, Russia

E-mail: [a]o.yakushkin@spbu.ru

The paper is focused on computational experiments on a system of continuous integration within the available infrastructure of MPD Root project. Test results of execution speed and its optimization options are presented for the builds in question.

The load of a computing node employed in continuous integration was analyzed in terms of performance of the central processor, RAM, and network connections. Various parameters of the build's parallel launch on different computing nodes were considered.

Thus, we substantially decreased the build time: from 45 to 2-3 minutes. The optimization was ensured by means of caching the project's dependencies and environmental components. Caching was done by Docker container manager.

Keywords: Docker, GitLab, CI, caching dependencies, MPD Root

The work was supported by RFBR research project № 16-07-011113 and SPBU projects 0.37.155.2014 and 9.37.157.2014

Introduction

This paper is focused on reducing software build time in a continuous integration system. It carries on our earlier study describing the development of an automated build system. In this paper, we consider the possibilities to speed up the operation of this system. We also put forward recommendations on its debugging.

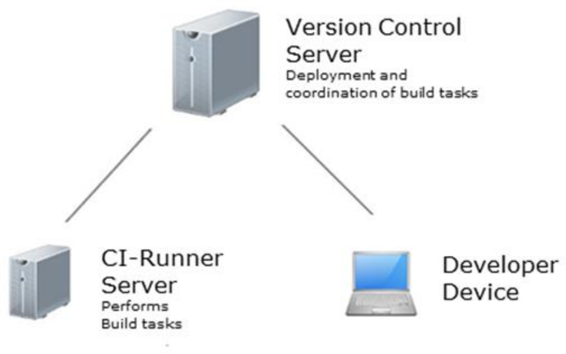
Automated building and testing allows a developer to track the entire build process with log files and identify the cause of an error. Log files provide data on the course of build process and the interaction between the system's components.

The time required by the build and test cycle is a crucial parameter of CI systems. The relevant feedback on the changes submitted to the project should be received by developers as soon as possible.

Automated build is especially vital when developing distributed systems and large-scale projects [Iakushkin2014, Degtyarev2014, Bogdanov2015, Shichkina2016]. The developers may simultaneously work on absolutely different build stages, which results in conflicting changes and complicates set-up and optimization [Ståhl2014, Iakushkin2016, Abrahamyan2016].

Problem statement

Our study aims to consider the parallel build options within a single GitLab CI node deployed on Windows Azure D16V2 platform. The architecture of the system in question is shown in Picture 1.



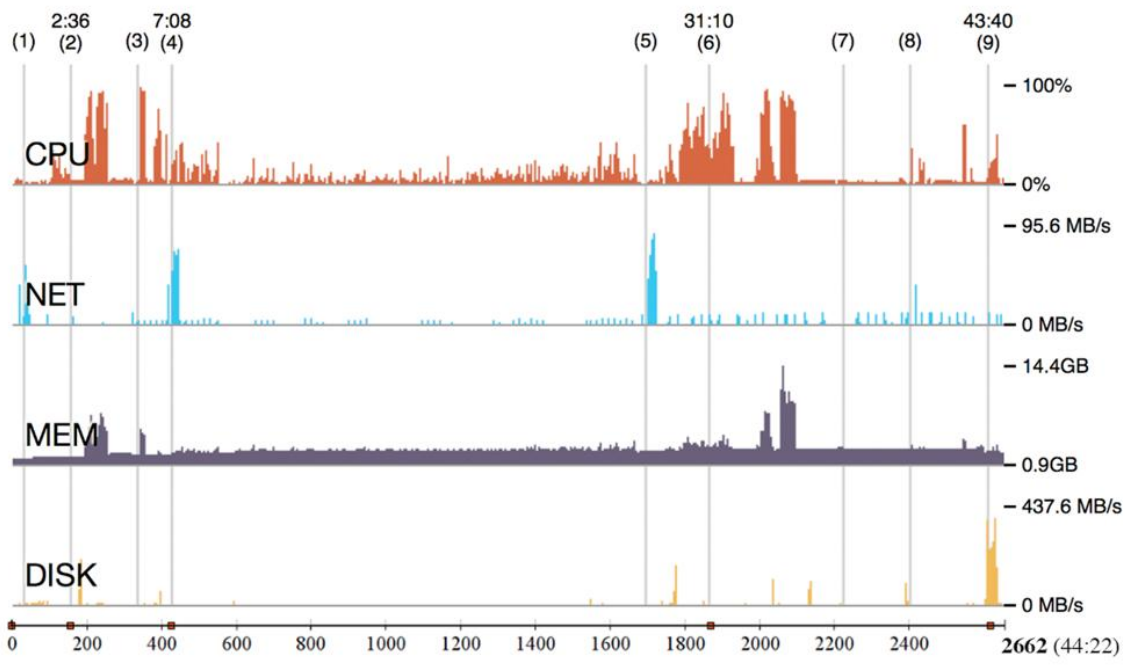
Picture 1: Hardware nodes of CI system

The version control server (the VCS) has the following parameters: CPU: Octa core Intel Xeon CPU E5-2673 v3 (-HT-MCP-) 2.40GHz; number of cores: 8; threads per core: 1; RAM: 28GB; SSD: 91.6GB; OS: Ubuntu 14.04. The parameters of CI server that distinguish it from the VCS: number of cores: 20; RAM: 140GB; SSD: 1TB.

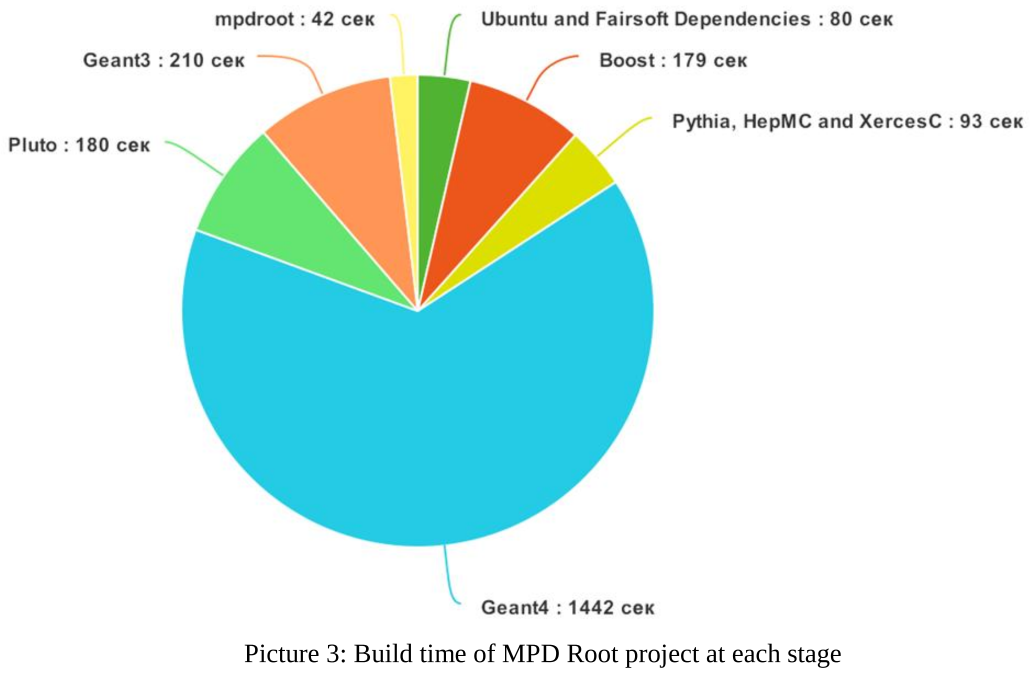
Experiments

We conducted a series of tests to identify the impact of dependencies caching and to evaluate the system's performance depending on the load and the amount of allocated resources.

Building from scratch



Picture 2: Histograms of CPU load, network load, RAM load, and disc load during full build of MPD Root (time in seconds).



Picture 3: Build time of MPD Root project at each stage

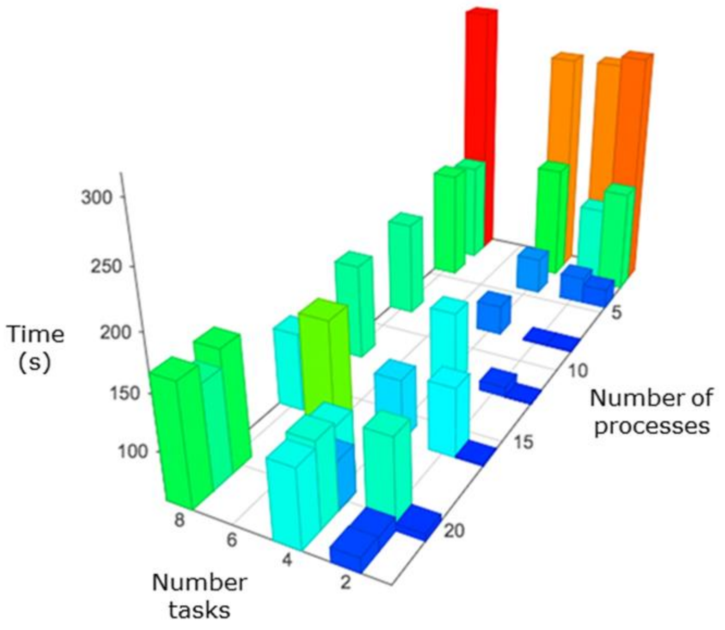
First, we made a step-by-step analysis of the build process from scratch. Picture 4 shows four histograms of system load during building. The parameters measured include CPU load, network load, RAM load, and disc load. The histograms show resource consumption during the build process. They also mark important stages and, in some instances, the time from the start of building to the onset of the next stage: (1)-(2) installation of Ubuntu and FairSoft dependencies; (2)-(3) installation of Boost library; (3)-(4) compilation of Pythia, HepMC and XercesC; (4)-(6) installation of Geant4; (5) down-load of data for Geant4 from the Internet; (6)-(7) compilation of Root; (7)-(8) build of Pluto; (8)-(9) installation of Geant3; (9) starting to build mpdroot.

The pie chart in Picture 3 shows that MPD dependencies consume the major share of build time. The building takes place in a container that must have all the required dependencies relevant to the build in question. That is why the reduction of build time cannot be attained simply by manual installation of dependencies on the server. We consider that caching the built dependencies in a Docker image is the way to ensure the minimum build time. To that end, we offer a solution that uses Docker to cache container images. The dependencies will be rebuilt from scratch only if their parameters are modified. If the parameters remain unchanged, every new build of the main project will run in a container produced from the image that already has all the software pre-installed. This approach allows to reduce the build time from over 45 min down to 2 minutes (i.e., more than by 20 time) for the second and the subsequent builds with the same dependencies.

In addition, stages 4-6 (build of Geant4) and 7-8 (build of Pluto) reveal a very low CPU load. Optimization of these packages' build process will allow to further reduce the duration of building from scratch.

Parallel build of MPDRoot

The next tests did not involve installation of dependencies. They aimed to ascertain the dependence between the build time of the main package upon the number of parallel build tasks and the number of allocated CPU threads. The results are shown in Picture 4:



Picture 4: The diagram showing the dependence of MPD Root build time upon the number of parallel builds and parallel threads. The minimum time of one build—56 seconds.

Picture 4 demonstrates that the system needs at least five threads, regardless of the number of tasks. Otherwise the build time will be unjustifiably long. The allocation of five or more threads allows to complete separate CI tasks in less than one minute. When one or more parallel tasks are added, the build takes three times more. It makes it necessary to allocate additional hardware resources or optimize the parallel use of the existing ones.

Conclusion

Shot build time is a major principle of CI systems. A developer should be able to check the system's functionality right after introducing the changes. This allows to fix bugs at an early stage. That is why the build time is the main parameter requiring optimization. In this paper, we analysed the ways to implement a parallel build and to optimize the build time.

Caching dependencies in complex projects has the most significant effect on build time reduction. We run tests that involved dependencies caching with Docker image layers, which resulted in build time reduction from 45 min down to 2-5 min.

Optimization of the project's parallel builds is vital when setting up a CI system, because builds take place often and can be run by many programmers. The analysis of the developed system's performance demonstrated that it is advisable to use more than five parallel threads when building the MPD Root project.

The future study is going to include tests on increasing the number of CI servers physically located on different machines. This will allow to execute more CI tasks concurrently while avoiding resource starvation.

References

Iakushkin, Oleg, and Valery Grishkin. "Unification of control in P2P communication middleware: Towards complex messaging patterns." PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON NUMERICAL ANALYSIS AND APPLIED MATHEMATICS 2014 (ICNAAM-2014). Vol. 1648. No. 1. AIP Publishing, 2015.

Iakushkin O.O., Degtyarev A.B., Shvemberger S.V. Decomposition of the modeling task of some objects of archeological research for processing in a distributed computer system // Computer Research and Modeling. — 2014. Vol 7, No. 3. – P. 533-537

Ståhl, Daniel, and Jan Bosch. "Modeling continuous integration practice differences in industry software development." Journal of Systems and Software 87 (2014): 48-59.

Iakushkin, Oleg, Yulia Shichkina, and Olga Sedova. "Petri Nets for Modelling of Message Passing Middleware in Cloud Computing Environments." In International Conference on Computational Science and Its Applications, pp. 390-402. Springer International Publishing, 2016.

Shichkina, Yulia, Alexander Degtyarev, Dmitry Gushchanskiy, and Oleg Iakushkin. "Application of Optimization of Parallel Algorithms to Queries in Relational Databases." In International Conference on Computational Science and Its Applications, pp. 366-378. Springer International Publishing, 2016.

Abrahamyan, Suren, Serob Balyan, Avetik Muradov, Vladimir Korkhov, Anna Moskvicheva, and Oleg Jakushkin. "Development of M-Health Software for People with Disabilities." In International Conference on Computational Science and Its Applications, pp. 468-479. Springer International Publishing, 2016.

Bogdanov, A., A. Degtyarev, V. Korkhov, V. Gaiduchok, and I. Gankevich. "Virtual supercomputer as basis of scientific computing." Horizons in Computer Science Research 11 (2015): 159-198.