plcli - a Tool for Running Distributed Applications on PlanetLab

Axel Niklasson
axelni@student.chalmers.se
Department of Computer Science and Engineering
Chalmers University of Technology, Sweden

Abstract

Implementing and testing distributed applications on platforms with many servers such as PlanetLab is made easier through the help of tooling. Various solutions exist, but most tools are either outdated or lack sufficient documentation. In this paper, a tool referred to as *plcli* is introduced which, among other things, enables engineers working with PlanetLab to deploy and run distributed applications on PlanetLab servers. The intended functionality of *plcli* has been summarised in three use cases; running distributed experiments, distributed debugging and node health monitoring. Furthermore, a pilot implementation written in the programming language Go is offered with this paper that implements support for experiment deployment. To see how *plcli* performs, an evaluation has been carried out that has shown that the deployment time is nearly kept constant as the number of application instances and servers are scaled up to as much as 120 instances on fifteen servers. For example, a 400% increase in the number of servers only resulted in a 7% increase in deployment time.

1 Introduction

When implementing and testing applications that are meant to run as distributed systems, the need for tooling to deploy these applications arise. In order to understand how the application in question performs as a real system, it must be deployed as such; while simulating a physically distributed system using tools such as NS-3 [1] [2] is a great alternative during the development phase, testing must be performed in a real-world environment as well. Testing applications for production usage often requires ten or more servers to be part of the deployments, which emphasises the importance of automating the process of deployment. Manually connecting to servers or residing to ad-hoc implemented scripts is not a scalable solution, which is what this kind of tooling aims to provide.

This paper focuses on applications and experiments run on the PlanetLab EU platform [3]. PlanetLab is a platform used for deploying, running and accessing distributed applications in a planetary-scale system [4] [5] [6]. More than 300 universities and research institutes, referred to as *sites*, are providing servers to the network. These servers are called *nodes* and are what makes out the computing capacity of the enormous cluster that is PlanetLab. Users are assigned a *slice*, which essentially is a distributed virtualised virtual machine, that they can use to deploy services to. Since nodes are provided by the different sites, no guarantees on homogeneity on the machines can be given and downtime is to be expected. The varying quality of nodes makes PlanetLab a suitable platform for testing systems in a real-world setting. However, users of this platform need to make sure that desired nodes are actually functioning as intended, which can at times be tedious work.

1.1 Related work

There are public user tools listed on the PlanetLab website that may be used to decrease workload and enable easier usage when working with PlanetLab [7]. The listed tools offer various functionalities such as slice management, package management and others. Most of the tools (8/12) are not available anymore and the list is gravely outdated, but two tools that are still accessible and exhibit similar functionality as *plcli* are *PLDeploy* and *pssh*. PLDeploy functions as a "utility to deploy, configure and control PlanetLab services" which is rather similar to Use Case 1, presented in Section 1.2. However, when deploying services using PLDeploy, it needs to be done in a very special fashion by constructing and attaching what is called *cogs* that are used to deploy services and pulling the results back. There is not much information related to this tool and its documentation has not been updated in the last decade, making a more in-depth comparison hard to perform. pssh on the other hand provides a parallel version of OpenSSH and related tools [8] and its main features of providing parallel execution of commands over ssh is also present in *plcli*. For example, when users want to run a command on several nodes, this is done in a concurrent fashion without the users having to consider it.

1.2 Use Cases

The first use case, **Use Case 1**, targets support for deploying an application to a given number of physical PlanetLab nodes, launching a specified amount of instances on each node and upon termination, gathering of log files. As outlined in Section 2, engineers using *plcli* are able to quickly deploy their experiments through adding a configuration file to their application repository. Use Case 1 is considered the main feature of *plcli* and is the only one implemented in the preliminary version of *plcli*.

Apart from running experiments, being able to carry out distributed debugging is also an attractive feature and is expressed as **Use Case 2**. Finding out what is happening in a distributed system is a very complex task, as clearly outlined by Joyce *et al.* [9] and support for distributed debugging could increase engineering productivity a great deal as well as aiding in finding out why certain problems occur in production systems. For example, an approach similar to D3S presented by Liu *et al.* in [10] or the solution based on the MINHA platform presented by Jorge *et al.* [11] could be taken to implement support for this use case.

The duration of experiments might range from a few minutes to hours or even days and it is important that these experiments are performed on healthy nodes. Features such as healthy node discovery and automatically fixing some of the more simple problems of nodes (for example problems that may be resolved through a simple reboot) could be added to *plcli* to provide a richer toolset when working with PlanetLab nodes, which is referred to as **Use Case 3**. This could be taken even further by considering the PlanetLab platform as a system in itself and deploy planning agents that make decisions based on repair plans and carry out node repairs, which is an approach heavily based on the one presented by Dashofy *et al.* [12].

1.3 Contribution

In this paper a tool for deploying and running distributed applications using the PlanetLab platform known as *plcli* has been introduced, along with three main use cases representing the three main features of the tool. A preliminary implementation of *plcli* is bundled with this paper which provides functionality for Use Case 1. An evaluation of the performance of the tool with respect to Use Case 1 has been carried as well, which is presented and discussed. An implementation satisfying Use Case 1 is offered alongside this paper [13].

2 System Design

plcli is a command-line interface written in the programming language Go, which is a programming language introduced by Google in 2009, designed for fast compilation of source code and easier programming [14] [15] [16]. The Go programming language provides functionality for implementing efficient applications with scalable concurrency mechanisms known as *goroutines*. Goroutines are essentially lightweight threads associated with less overhead and the Go runtime is very efficient in the handling of these goroutines. As shown by Togashi *et al.*, Go outperforms for example Java when it comes to concurrency handling [17]. Mechanisms for efficient concurrency handling are naturally of high interest when developing a tool such as *plcli* that must be able to communicate with tens, or even hundreds, of nodes efficiently. These goroutines are further referred to as *workers* and are used whenever there are I/O-bound operations that need to be executed concurrently, such as calling the PlanetLab API or executing commands on nodes over SSH.

plcli is a preliminary implementation of a full-fledged tool intended to provide support for all three use cases outlined in Section 1.2. Through the implementation of support for Use Case 1, various features have been added to *plcli* such as deployment of applications, file transfer to nodes as well as concurrent command execution. An extensive list of available commands at the time of writing can be found in the README in the project repository [13].

Since the main functionality of the current implementation is to deploy code, a more in-depth explanation of how a deployment is performed is provided. *plcli* makes use of what is called a configuration file which is required to be placed in the root of the public repository of applications that should be deployed using *plcli*. This file contains information about environment variables, how the application is prepared for launch and how to launch an instance. *plcli* downloads this file from the application repository and performs the needed steps in order to prepare the PlanetLab nodes for deployment of the given application. The structure of this file is not finalised at the time of writing and omitted for brevity. However, an example can be found in the GitHub repository for the demo application [18].

plcli aims to reduce the time and effort needed to deploy and run experiments on PlanetLab and consequently, it is highly dependant on the performance of the PlanetLab API. In order to retrieve information about what nodes are available and decide which nodes to use in a deployment, the API is queried for up to date information. However, the API is only used internally by *plcli* in an effort to enable users and researchers to focus on what experiments to run rather than how these experiments actually are run.

3 Evaluation Plan

In order to investigate the performance of *plcli*, the time taken to perform a full deployment - the *deployment time* - is evaluated in three different experiments. All three experiments presented below utilise a basic demo application written in Python 3.7, which can be found on GitHub [18] and were run on a MacBook Pro 15" with a 2,2 GHz Intel Core i7 processor and 16 GB of RAM.

In the first experiment, one application instance was deployed to a varying amount of physical nodes with one worker allocated per node. This was done to investigate changes in deployment time as the deployment grows with respect to the number of nodes. Furthermore, experiments with one instance deployed to a fixed amount of nodes with a varying amount of workers were also conducted, which aided in investigating the performance gain of using workers. Lastly, due to severe problems with finding more than fifteen suitable nodes for the demo application, a varying amount of application instances were launched on the same set of physical nodes to try and investigate the performance at scale.

4 Evaluation Results

The first experiment measuring the performance of *plcli* when deploying to an increasing amount of physical nodes exhibits a nearly constant deployment time, as can be seen in Figure 1. There is a small increase of the trendline as the number of nodes increase, but it is very small and indicates that the overhead introduced by adding more nodes than fifteen will be minimal. For example, the time taken to deploy to three nodes is around fifteen seconds while deploying to fifteen nodes takes around sixteen seconds; that is a 400% increase in the number of nodes with merely 7% increase in deployment time which indicates promising performance when scaling. These results are expected, since one worker is allocated per instance and consequently a lot of work can be carried out in an efficient fashion.

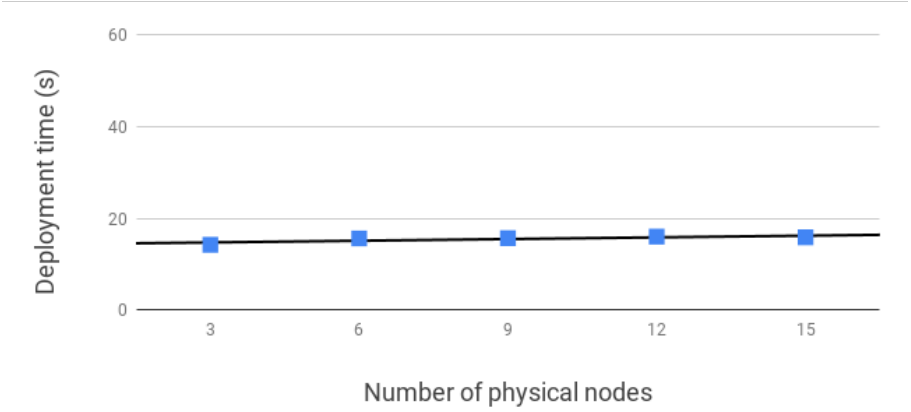

Figure 1: Deploying one instance to an increasing number of physical nodes with one worker per instance.

The results from the second experiment, investigating the effect of workers with respect to the deployment time, can be seen in Figure 2. As can be seen, the deployment time is nearly cut in half as the number of workers are doubled. The reason for the time being a bit unevenly reduced as the number of workers are doubled is most likely due to network latency and other operations that are subject to variation in execution time. The reason for not deploying using more than twelve workers is that since twelve physical nodes are used, more than twelve workers would not make any sense since there would not be any work for the additional workers. These results are expected, since in theory, a 100% increase in workers should yield a 50% decrease in execution time since there are twice as many workers available to perform the deployments.

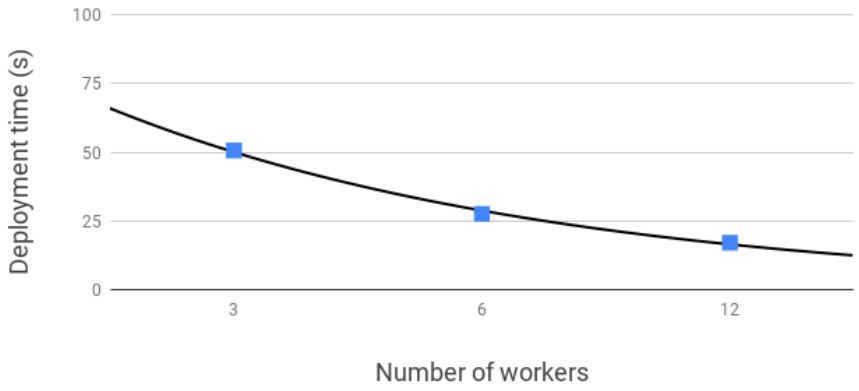

Figure 2: Deploying one instance to twelve nodes with an increasing number of workers.

Results from the third experiment, examining the change in deployment time when increasing the amount of instances on a constant amount of nodes, can be seen in Figure 3. It shows the increase of deployment time as the number of instances is scaled up to as much as 120 instances launched on fifteen physical nodes. Much similar to Figure 1, the deployment time is slowly growing but almost kept constant as the number of instances is multiplied by 8x. Due to one worker being used per instance launch, an almost constant deployment time is to be expected.

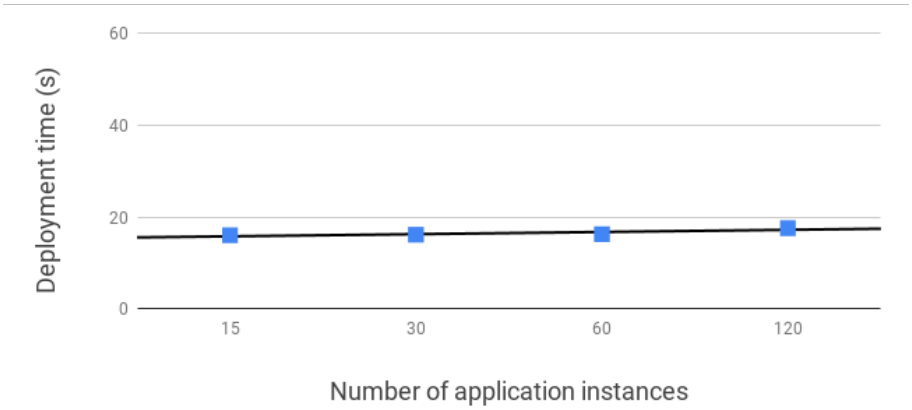

Figure 3: Scaling deployments virtually with one worker per instance to fifteen physical nodes.

5 Conclusion

A first implementation of *plcli* supporting Use Case 1 has been implemented and shown to provide a nearly constant deployment time for up to as much as 120 application instances on 15 physical nodes on the PlanetLab platform. The approach of using workers has been shown to be directly linked to the deployment time and shown to be very beneficial when performing concurrent deployments. Furthermore, *plcli* is bundled with many useful features for engineers using PlanetLab to run applications. When it comes to further work, *plcli* would benefit greatly from implementation of Use Case 2 and Use Case 3, as well as a more rigorous evaluation with respect to scalability (i.e. deploying to more physical nodes). This preliminary implementation is to be considered as a pilot and a foundation for continued work.

References

[1] ns-3 — a discrete-event network simulator for internet systems. https://www.nsnam.org/. Accessed; 2019-07-12.

[2] Teerawat Issariyakul and Ekram Hossain. *Introduction to Network Simulator 2 (NS2)*, pages 1–18. Springer US, Boston, MA, 2009.

[3] Planetlabeurope. https://www.planet-lab.eu/. Accessed; 2019-07-12.

[4] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. Planetlab: An overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, 33(3):3–12, July 2003.

[5] Larry Peterson and Timothy Roscoe. The design principles of planetlab. *SIGOPS Oper. Syst. Rev.*, 40(1):11–16, January 2006.

[6] Larry Peterson, Andy Bavier, Marc E. Fiuczynski, and Steve Muir. Experiences building planetlab. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, pages 351–366, Berkeley, CA, USA, 2006. USENIX Association.

[7] User tools. https://www.planet-lab.org/tools. Accessed; 2019-09-14.

[8] parallel-ssh. https://code.google.com/archive/p/parallel-ssh/. Accessed: 2019-07-03.

[9] Jeffrey Joyce, Greg Lomow, Konrad Slind, and Brian Unger. Monitoring distributed systems. *ACM Trans. Comput. Syst.*, 5(2):121–150, March 1987.

[10] Xuezheng Liu, Zhenyu Guo, Xi Wang, Feibo Chen, Xiaochen Lian, Jian Tang, Ming Wu, M Frans Kaashoek, and Zheng Zhang. Ds: Debugging deployed distributed systems. pages 423–437, 01 2008.

[11] Tiago Jorge, Francisco Maia, Miguel Matos, José Pereira, and Rui Oliveira. Practical evaluation of large scale applications. In Alysson Bessani and Sara Bouchenak, editors, *Distributed Applications and Interoperable Systems*, pages 124–137, Cham, 2015. Springer International Publishing.

[12] Eric M. Dashofy, André van der Hoek, and Richard N. Taylor. Towards architecture-based self-healing systems. In *Proceedings of the First Workshop on Self-healing Systems*, WOSS '02, pages 21–26, New York, NY, USA, 2002. ACM.

[13] plcli github repository. https://github.com/axelniklasson/plcli. Accessed; 2019-09-14.

[14] The go programming language. https://golang.org/. Accessed; 2019-07-09.

[15] Rob Pike. The go programming language. *Talk given at Google's Tech Talks*, 2009.

[16] Alan AA Donovan and Brian W Kernighan. *The Go programming language*. Addison-Wesley Professional, 2015.

[17] N. Togashi and V. Klyuev. Concurrency in go and java: Performance analysis. In *2014 4th IEEE International Conference on Information Science and Technology*, pages 213–216, April 2014.

[18] plcli demo app github repository. https://github.com/axelniklasson/plcli-demo-app. Accessed; 2019-09-14.

A Existing PlanetLab tools

Tables 1 and 2 list all the tools listed on the PlanetLab website[1] as of July 2, 2019.

Name	Brief description	State
Plush	Users describe experiments or computation in XML, and Plush uses it to locate, contact, and prepare resources. It includes a Nebula GUI that allows users to build, visualize and run their applications without using the command-line interface.	Can't access webpage.
PIMan	PlanetLab Experiment Manager is designed to simplify the deployment, execution and monitoring of your PlanetLab experiment. The application presents a simple GUI to perform common tasks.	Can access webpage, but all links are broken.
Stork	A software installation utility akin to yum and apt available for both users of PlanetLab and for home use. It includes a Stock Slice Manager GUI that simplifies package management and Stork installation on your PlanetLab slices.	Broken link.
pShell	A Linux shell like interface providing a few basic commands to interact with a Planetlab slice, works as a command center at the local machine and interact with slice nodes.	Broken link.
AppManager	PlanetLab Application Manager is designed to help deploy, monitor, and run applications on PlanetLab. The package gives you the ability to centrally manage, install, upgrade, start, stop, and monitor of applications on a PlanetLab slice.	Broken link.

Table 1: Tools listed on the PlanetLab website and their state as of July 2, 2019 (Table 1/2)

[1] https://www.planet-lab.org/tools

Name	Brief description	State
Emulab	A network testbed, giving researchers a wide range of environments in which to develop, debug, and evaluate their systems.	Accessible, but different purpose than *plcli*.
plDist	A tool for parallel distribution of files to Planetlab nodes using BitTorrent or rsync.	Broken link.
Nixes	A set of bash scripts to install, maintain, control and monitor applications on PlanetLab.	Broken link.
PLDeploy	PlanetLab Slice Deploy Toolkit is a set of scripts to help users manage their slices.	Accessible, comparison with *plcli* can be found in Section 1.1.
pssh	Provides the parallel versions of the openssh tools. It can be used to control large collections of nodes in the wide-area network.	Accessible, comparison with *plcli* can be found in Section 1.1.
vxargs	Inspired by xargs and pssh, it provides the parallel versions of any arbitrary command, including ssh, rsync, scp, wget, curl, etc.	Broken link.
PlanetLab broadband link emulator	A link emulator for PlanetLab that can be configured with few important measured characteristics of broadband links, such as their asymmetric link bandwidths and queue sizes.	Accessible, but different purpose than *plcli*.

Table 2: Tools listed on the PlanetLab website and their state as of July 2, 2019 (Table 2/2)