P2P experimentations with SPLAY: from idea to deployment results in 30 min.*

Lorenzo Leonini† Étienne Rivière‡ Pascal Felber
University of Neuchâtel, Switzerland

Abstract

SPLAY is an integrated system that facilitates the complete chain of distributed systems evaluation, from design and implementation to deployment and experiments control. Algorithms are expressed in a concise, yet very efficient, language based on Lua. Implementations in SPLAY are highly similar to the pseudo-code usually found in research papers. SPLAY eases the use of any kind of testbeds, e.g., PlanetLab, ModelNet clusters, or non-dedicated platforms such as networks of workstations. Using SPLAY and PlanetLab, this demonstration highlights a complete evaluation chain of an epidemic protocol and a churn-driven experiment using the Pastry DHT.

1 Introduction

Evaluating large-scale distributed applications is a higly complex, time-consuming and error-prone task. One of the main difficulties stems from the lack of appropriate tools for quickly prototyping, deploying and evaluating algorithms in real settings. Several dedicated testbeds are available that can be leveraged for better evaluations of these systems: PlanetLab [2], Everlab [8], or network emulators such as ModelNet [13] or Emulab [15]. Meanwhile, non dedicated testbeds such as networks of idle workstations usually found in research labs or schools, are difficult to use for distributed systems experiments, as one usually require access rights that are not easily granted by the admistrators.

All these testbeds are appealing as they allow real or realistic experiments to be conducted, but they are not used as systematically as they should. Indeed, strong technical skills are typically necessary to develop, deploy, execute and monitor applications for such testbeds. The learning curve is also usually slow. Technical difficulties are even higher if one wants to deploy an experiment on several testbeds at the same time, for instance a population of peers on adversial testbeds such as PlanetLab and another population of peers on a local ModelNet cluster.

*This work is supported in part by the Swiss National Foundation under agreement number 102819.

†Contact author: `lorenzo.leonini@unine.ch`

‡This work was carried out during the tenure of an ERCIM "Alain Bensoussan" Fellowship Programme.

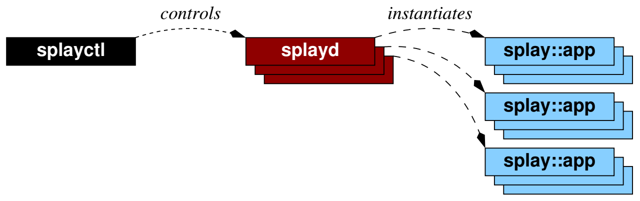
A side effect of these difficulties is that the performance of evaluated systems is greatly impacted by the technical quality of their implementation, overshadowing the underlying algorithm's intrinsic qualities. This may in turn make comparisons unsound or irrelevant.

All these observations call for novel development-deployment systems that would straightforwardly exploit these testbeds and bridge the gap between algorithmic specifications and live systems. Researchers could use such a system for real evaluations instead of simulations, and teachers to focus their lab work on the core of distributed programming, i.e., algorithms and protocols, letting students experience distributed systems implementation in real settings.

Related work. There were a number of proposals in the litterature for evaluation or deployment frameworks. The former includes systems such as Mace [9] (C++ extension) or P2 [10] (dedicated declarative language) that allow developers to express algorithms in a high-level language, hiding most of the complexity. These languages, however, do not provide any support for deploying the applications or controling the behavior of an execution. The latter include deployment tools such as Plush [1] or Weevil [14]. Both allows the creation of deployment scripts for testbeds such as PlanetLab. Using the user's description of the experiment, they instantiate the applications on the testbed or create scripts for this task. Nonetheless, these systems do not allow deployment on non-dedicated testbeds. They do not allow either for complex deployment, for instance involving multiple testbeds or complex network scenarios, that SPLAY allows natively and without efforts.

2 SPLAY architecture

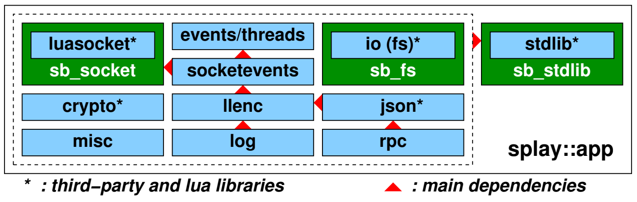
A SPLAY infrastructure is composed of a controller, daemons and sandboxed application processes:



The controller is a trusted entity that manages the deployment and execution of SPLAY application. Lightweight daemons processes (splayd) are responsible for instantiating sandboxed applications processes, as instructed by the controller. A splayd can run multiple sandboxed application instances. Sandboxing is a primary feature of SPLAY; it allows the administrator who deploy the daemons to restrict the usage of local resources (memory, disk, network). Applications instances have absolutely no direct access to the hosting system.

Language and libraries. SPLAY is based on Lua [7], a highly efficient scripting language. A dedicated language is needed for several reasons. First, as we need to support non dedicated testbeds, sandboxing is a sound basis. Lua support for scoping and first-order functions allows us to redefine all standard library functions to impose boundaries on resource usage. This allows us to ensure that buggy or ill-behaved code will not harm the system hosting the application. Second, it is necessary to support a large number of application processes on a single host. Our tests have shown that Lua for SPLAY is able to run more than 1,250 instances of Pastry [11] on a single dual-core machine with 2GB of memory. The others reasons include the possibility to run applications on any hardware or host OS, and the performance of the libraries.

We provide an extensive set of libraries for developping distributed applications with SPLAY, including: networking libraries with sandboxed RPCs and message passing with UDP or TCP (including automatic serialization); a sandboxed virtual filesystem; threading based on coroutines and event-based programming; a logging library to seamlessly report statistics about running applications.



An important goal of SPLAY is to allow application developers to write concise, readable code that highly resemble pseudo-code found in research papers. During this demonstration, we will code a self-contained epidemic diffusion protocol. We did implement a set of distributed systems using SPLAY. The Chord [12] and Pastry [11] DHTs are respectively 101 and 265 lines (including fault tolerance and initialization). Middleware using Pastry such as Scribe [5] and Splitstream [6] are 79 and 58 lines, respectively.

Controlling deployments. SPLAY provides either a command line or Web-based interface. It allows to select the daemons that will host an experiment based on geographical location, performance, load, resource limitations, etc. An interesting feature is the churn management module. To allow fair comparison of systems under the same conditions, and since the natural churn in PlanetLab is not always sufficient to derive a protocol's behavior, SPLAY can reproduce the dynamics of a system, either from a synthetic description (for instance, creating massive churn, steady increase, etc.) or from a real trace (e.g., [3]).

3 Demo overview

This demo will present an overview of the key components of SPLAY, from development to deployment. We will develop interactively, and within minutes, an epidemic diffusion protocol [4], highlighting key libraries and features. Using both command-line and Web-based interfaces, we will explore several node selection criterias. We will then deploy the live protocol on PlanetLab, and process the results, which we will compare with results from [4]. This complete tool chain will help to illustrate the simplicity of P2P systems evaluation permitted by SPLAY, and that we believe to be of great interest to the community. Finally, using a complete, fault-tolerant implementation of Pastry [11] we will present the use of one of the high end features of SPLAY, churn management. We will run Pastry on PlanetLab using the trace from [3], and observe key results such as distribution of delays and route failure ratios. SPLAY is available at http://www.splay-project.org. This website provides facilities for trying SPLAY, including an access to a PlanetLab deployment of SPLAY, a non-restricted *"SplayGround"* emulating a network of 1,000 machines on a single host, and offers to download a live CD that allows to try SPLAY without installing anything.

References

[1] J. Albrecht, et al. Remote Control: Distributed Application Configuration, Management, and Visualization with Plush. In *LISA'07*.
[2] A. Bavier, et al. Operating system support for planetary-scale network services. In *NSDI'04*.
[3] R. Bhagwan, et al. Understanding availability. In *IPTPS'03*.
[4] K. P. Birman, et al. Bimodal multicast. *ACM TOCS*, 17(2):41, 1999.
[5] M. Castro, et al. SCRIBE: A large-scale and decentralized publish-subscribe infrastructure. *IEEE J. Sel. Areas Commun.*, 2002.
[6] M. Castro, et al. SplitStream: High-bandwidth multicast in a cooperative environment. In *Proc. of 19th SOSP*, October 2003.
[7] R. Ierusalimschy, et al. The Implementation of Lua 5.0. *Journal of Universal Computer Science*, 11(7):1159, 2005.
[8] E. Jaffe, et al. Everlab: a production platform for research in network experimentation and computation. In *LISA'07*, pages 1–11.
[9] C. E. Killian, et al. Mace: language support for building distributed systems. In *Proc. of PLDI '07*, pages 179–188.
[10] B. T. Loo, et al. Implementing declarative overlays. In *SOSP'05*.
[11] A. Rowstron et al. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Middleware'01*.
[12] I. Stoica, et al. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 11(1):17, 2003.
[13] A. Vahdat, et al. Scalability and accuracy in a large-scale network emulator. In *Proc. of OSDI*, pages 271–284, 2002.
[14] Y. Wang, et al. Automating Experimentation on Distributed Testbeds. In *Proc. 20th ASE*. Long Beach, California, November 2005.
[15] B. White, et al. An Integrated Experimental Environment for Distributed Systems and Networks. In *Proc. of OSDI*, 2002.