Scalable Parallel Scripting in the Cloud

Benjamin Glick
Lewis & Clark College
Chicago, Il
glick@lclark.edu

Yadu Babuji (Advisor)
Computation Institute, University of Chicago/Argonne National Laboratory
Chicago, Il
yadunand@uchicago.edu

Kyle Chard (Advisor)
Computation Institute, University of Chicago/Argonne National Laboratory
Chicago, Il
chard@uchicago.edu

ABSTRACT

Parsl is a parallel scripting library for Python that provides a simple model for describing and executing dataflow-based scripts over arbitrary execution resources such clusters, grids, and high-performance systems. Parsl's execution layer abstracts the differences between providers enabling provisioning and management of compute nodes. In this poster we describe the development of a new execution provider for Parsl that is designed to support Amazon Web Services (AWS) and Microsoft Azure. This provider supports the transparent execution of implicitly parallel Python-based scripts using elastic cloud resources. We demonstrate that Parsl is capable of executing thousands of applications per second over this elastic execution fabric.

CCS CONCEPTS

• **Computer systems organization → Cloud computing**; *Distributed architectures*; Client-server architectures; • **Software and its engineering → Parallel programming languages**;

KEYWORDS

Cloud Computing, Python, High-Performance Computing, Parallel Scripting

ACM Reference Format:
Benjamin Glick, Yadu Babuji (Advisor), and Kyle Chard (Advisor). 2017. Scalable Parallel Scripting in the Cloud. In *Proceedings of The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC17), Denver, Colorado USA, November 2017 (SC'17),* 2 pages.

1 INTRODUCTION

Parsl is an open-source parallel scripting library for Python [2]. Rather than explicitly implement directed acyclic graphs or workflows, Parsl allows developers to "mark-up" standard Python scripts with decorators around Python functions (called Parsl "Apps"). These wrappers enable Parsl to transparently enable asynchronous and parallel execution of Apps.

Parsl scripts can be executed on any parallel system, from multicore computers through to supercomputers, without modification. In this poster we describe how we have extended Parsl to support parallel execution on two elastic cloud platforms: Amazon Web Services (AWS) Elastic Compute Cloud (EC2) and Azure. We describe our cloud execution model and explore its scalability and throughput.

SC'17, November 2017, Denver, Colorado USA
2017.

2 PARSL EXECUTION MODEL

Parsl's ability to transparently support different execution environments is based on its standard executor interface. The basic Parsl execution model is as follows. Execution of a Parsl App is called a *job*. The interface through which jobs are submitted to an *execution provider* (e.g., supercomputer) is called the *executor*. Parsl supports various executors to make use of execution resources, for example IPython Parallel (IPP) provides a general pilot model [4] and Swift/T enables extreme-scale, many task execution [5]. Configuration of an execution provider is based on a *site definition* file which describes what interface is available, what information is required for job submission, and what execution models are supported.

The execution provider interface consists of three functions: submit, status, and cancel. Submit is passed a job, executes it asynchronously, and returns a unique id. Cancel terminates a job given its job id and deallocates the resources used. Status returns a list of jobs and their statuses.

3 CLOUD EXECUTION PROVIDER

Figure 1 illustrates Parsl's cloud execution model. The cloud execution provider dynamically provisions, scales, and reuses an elastic pool of cloud computing resources. The provider uses the boto3 [1] and Azure Python SDK [3] libraries to create a virtual private cloud (VPC) on AWS and a virtual network on Azure, with associated internet gateway, for housing the execution instances. The provider is configured with AWS/Azure keys to securely access cloud APIs. Automated scripts are used to configure execution nodes with IPP to enable the execution of Parsl jos. The provisioning process follows best practices, ensuring appropriate security groups are configured to block incoming connections while still enabling outgoing and inter-node communication.

When Parsl determines that new workers are needed to execute jobs, the EC2 provider is used to provision instances in a preconfigured VPC. The provider uses EC2 user data, a cloud-init based startup script to install and start an IPP engine and to subsequently connect this engine to the IPP controller that is deployed on the local machine [4]. Any computer can act as the Parsl host as long as there is a persistent (and accessible) IPP controller hosted on that machine and the registration location and port are specified to the execution provider. After the instance is connected to the controller, Parsl jobs (when ready) are dynamically passed to it for execution. After all jobs are completed the entire execution infrastructure is removed and deallocated.

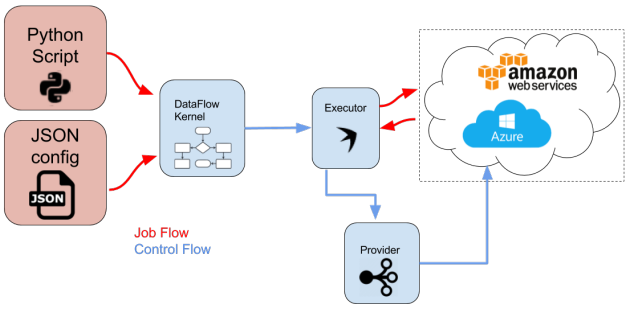

Figure 1: Parsl Execution Architecture

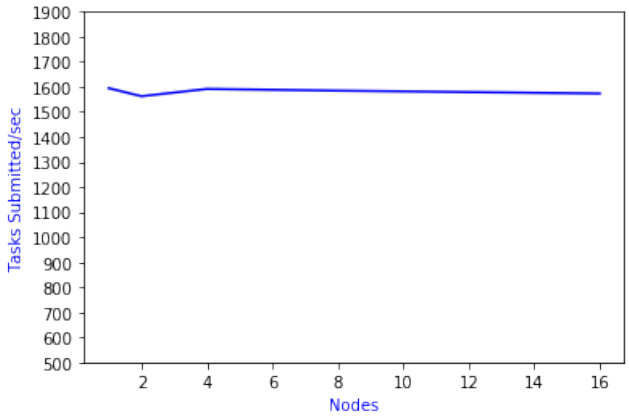

Figure 2: Job submission rate with different cluster sizes for 2 million Monte Carlo simulations of pi.

4 EVALUATION

To evaluate the performance of our cloud-based execution provider we deployed c4.large instances (2 intel xeon cpus and 4gb of memory) in clusters of up to 16 nodes.

We first investigated job submission rates as we scaled the cluster from 1 to 16 nodes. Figure 2 shows that the job submission rate remains constant with increasing cluster size. We were able to launch approximately 1500 jobs per second.

We then investigated cluster scalability. To do so, we ran a monte carlo simulation to compute pi. Figure 3 shows the results of running 1,000,000 total simulations. When run sequentially, the computation took approximately 4 seconds, when run on 16 nodes, it took approximately 250 milliseconds.

5 CONCLUSION & FUTURE WORK

The Parsl cloud executor enable users to rapidly scale applications across elastically scalable computing infrastructure with little knowledge of cloud or parallel computing. This capability further enhances Parsl's ability to democratize development of large-scale applications via its intuitive (Python-based) programing model and

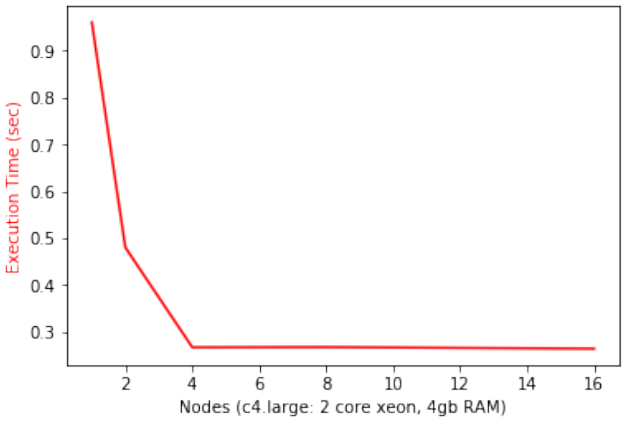

Figure 3: Execution time with different cluster sizes for 1 million Monte Carlo simulations of pi.

support for transparent use of heterogeneous execution providers. In future work we plan to expand the execution provider framework to support new execution providers, including the Google compute cloud and a number of leadership computing systems.

A ACKNOWLEDGMENTS

The authors would like to thank Yulie Zamora and Tyler Skluzacek for their guidance and support.

REFERENCES

[1] Amazon.com, Inc. Boto 3 documentation. https://boto3.readthedocs.io/en/latest/, 2014.
[2] Y. Babuji, A. Brizius, K. Chard, I. Foster, D. S. Katz, M. Wilde, and J. Wozniak. Introducing Parsl: A Python Parallel Scripting Library, Aug. 2017.
[3] Microsoft. Azure sdk for python. https://azure-sdk-for-python.readthedocs.io/en/latest/, 2016.
[4] The IPython Development Team. Using ipython for parallel computing. https://ipyparallel.readthedocs.io/en/latest/, 2015.
[5] J. M. Wozniak, T. G. Armstrong, M. Wilde, D. Katz, E. Lusk, and I. T. Foster. Swift/t: Large-scale application composition via distributed-memory data flow processing. http://swift-lang.org/papers/pdfs/Swift_2013.pdf, 2013.