Design and Architecture of a Gateway for Supporting Both Batch and Interactive Computing Modes on Supercomputers

Marjo Poindexter
Texas Advanced Computing Center
University of Texas at Austin
Austin, TX, USA
mpoindexter@tacc.utexas.edu

Rion Dooley
Texas Advanced Computing Center
University of Texas at Austin
Austin, TX, USA
dooley@tacc.utexas.edu

Joe Stubbs
Texas Advanced Computing Center
University of Texas at Austin
Austin, TX, USA
jstubbs@tacc.utexas.edu

Ritu Arora
Texas Advanced Computing Center
University of Texas at Austin
Austin, TX, USA
rauta@tacc.utexas.edu

Julia Looney
Texas Advanced Computing Center
University of Texas at Austin
Austin, TX, USA
jlooney@tacc.utexas.edu

***Abstract*— Supercomputing resources are used in both interactive and batch computing modes. The interactive mode is typically used for software development and testing purposes, or for running various command-line tools that work in a user-guided manner. The batch mode is often used for running a large number of jobs simultaneously or for large-scale runs that do not require interaction with the users. We *needed a gateway (a web-portal) that could support both interactive and batch computing modes*. The interactive mode was required for using a command-line tool for code parallelization through a web browser, and the batch mode was required for compiling and running the programs parallelized using the tool on the production systems at open-science data centers. We are iteratively developing this gateway, and its current version is live at *https://ipt.tacc.cloud*. An overview of the design and implementation of this gateway is presented in this paper. With minimal modifications, the architecture of this gateway can be reused for supporting other similar projects.**

Keywords—interactive mode, batch mode, supercomputing, Agave, Abaco, Interactive Parallelization Tool (IPT), SGCI, Gateway, Serverless, Functions-as-a-Service (FaaS), HPC

I. Introduction

The Interactive Parallelization Tool (IPT) is a command-line tool for parallelizing serial C/C++ applications in a user-guided manner [1]. The parallel programming models that are currently supported by IPT are MPI [2], OpenMP [3] and CUDA [4]. During execution, IPT *interactively* solicits specifications from the end user regarding what to parallelize and where, eventually generating a parallel version of the code. It is very natural to compile the parallel versions of the code generated by IPT and test them for accuracy and performance by running them on the High Performance Computing (HPC) platforms (or supercomputers). While small-scale testing can be done directly on the compute nodes on which IPT is running, for the large-scale runs, users may find that the HPC platforms of their choice are oversubscribed at a given point in time and hence, they may face a significant wait-time when attempting to run their code. In these situations, *batch* job execution through the existing scheduling system is used. In this manner, both interactive and batch computing paradigms are required throughout the process of developing parallel programs using IPT and in testing these programs on the production-quality HPC platforms.

One of our goals was to lower the adoption barriers to IPT by making it available in a ready-to-use and secure manner. To meet this goal, we opted for deploying IPT in the cloud and making it accessible via a web browser so that the users do not need to download any additional software for using IPT. Another goal was to provide a unified interface for supporting the compilation and testing of the code generated by IPT on production-quality HPC resources. The effort to materialize these goals has culminated into the IPT gateway: *https://ipt.tacc.cloud*. This gateway supports the entire workflow for interactively generating parallel code using IPT (Figure 1), and compiling and testing the generated code on remote HPC platforms in batch mode.

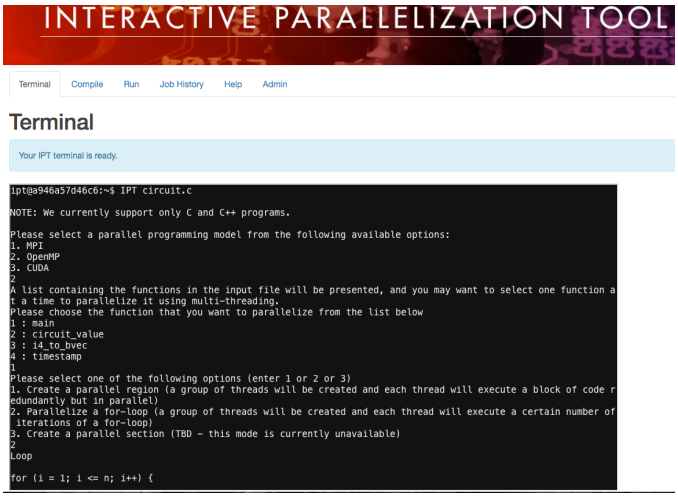

Figure 1. Terminal supported in the IPT gateway

The IPT gateway was designed to optimally use the existing cloud computing resources and services that are available to the open-science community. It was implemented with the support from the National Science Foundation (NSF) funded Science Gateway Community Institute (SGCI) [5]. The gateway further leverages the investments made by NSF in the

XXX-X-XXXX-XXXX-X/XX/$XX.00 ©20XX IEEE

form of the Agave [6] and Abaco [7] software projects. The job submissions through the IPT gateway are faciliated through an XSEDE [8] community account and a local community account. The IPT gateway is deployed on the NSF funded Jetstream Cloud [8] and the VMware system that is operated by the Texas Advanced Computing Center (TACC). It supports job submission and execution on TACC's Stampede2 [8] and Lonestar5 [9] HPC clusters, and San Diego Supercomputing Center's Comet [8].

A high-level description of the IPT gateway architecture is presented in Section 2 followed by a brief summary of the user interface considerations in Section 3. Section 4 discusses several of the expected and unexpected returns on investment from reusing existing platforms and technologies while section 5 introduces some related technologies and the reasons for choosing our current stack.

II. IPT Gateway Architecture

The IPT gateway is a combination of (1) a stateless Django web application, Abaco, Docker Swarm, TACC's instance of the Agave Platform, and HPC and storage allotments through community allocations from XSEDE and TACC. Each of these components is described in detail in the subsequent sub-sections. The architecture of IPT gateway is shown in Figure 2.

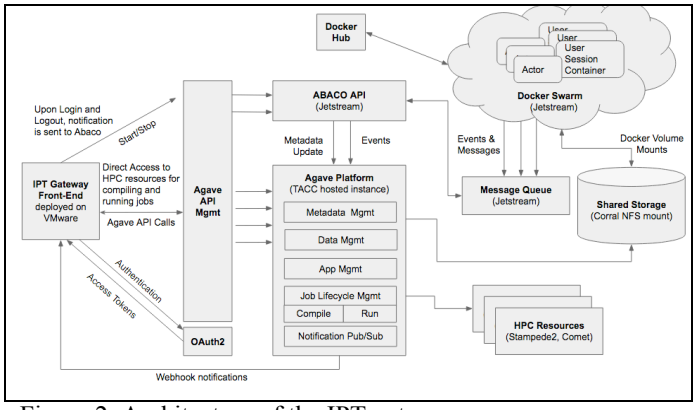

Figure 2. Architecture of the IPT gateway

A. IPT Gateway Application

IPT gateway is built upon the Django web application framework [10]. It leverages Agave's OAuth2 server for user authentication and relies on proxied calls to Agave for job, connection, data, and historical information. IPT gateway uses the Agavepy [11] Python library for all of its interactions with Abaco and Agave. While existing frontends are readily available for Agave, we chose to use Django because it provides a natural extension point from which an administrative backend dashboard can be built for managing and monitoring user sessions, generating reports, and controlling access to the gateway.

B. The Agave Platform

Agave is an open source, Science-as-a-Service (SaaS) platform for the open-science community. While the Agave Platform provides a broad range of functionality, for the purposes of the IPT gateway, Agave provides a high-level HTTP-friendly API for gateway developers to interact with different systems and applications to run code, manage data, collaborate with others, and integrate third-party services. Agave provides history in a way that enables repeatability, and a permissions model for keeping applications and results private, sharing with select colleagues, or publishing to entire communities. The Agave software provides the API for managing data movement (file-upload and download) and authorization. It keeps track of the metadata for terminal instances (used in the IPT gateway) such as the URL, username, and status of the instances.

C. Actor-based Computing (Abaco)

Abaco provides a mechanism for asynchronous processing in gateways (*e.g.*, to start up some long running process in the background). It provides all the scheduling, monitoring, logging, security, *etc.* needed to realize a serverless platform while also ensuring consistency of user data across sessions through enforcement of predefined policies around data volume accessibility and portability.

In the IPT gateway, Abaco manages the Docker containers running the terminal emulator and IPT executable. Starting and stopping these containers as part of user login/logout takes too long to do synchronously within a web request, so it must be queued in the background. Docker containers for the IPT gateway can do this by making an API request to Abaco. From the administrator console in the IPT gateway, one can stop all the containers (for different users) by clicking on stop buttons provided on the web-interface without waiting for one container to stop and going to another one. This functionality is also facilitated by Abaco. The metadata related to the status of the containers (start/stop states) and users is also tracked with Abaco.

D. Interaction of the various software components

When a user logs onto the IPT gateway, and requests a terminal, the Django application calls Abaco's REST API requesting a session for the user. The IPT gateway tracks user sessions internally using Agave's Metadata service, while Abaco manages session containers and metadata. This separation of concern allows independent scaling of the frontend and backend components.

When Abaco receives a new session request, it communicates to the Docker Swarm cluster to launch a new agent container for the user. The agent then starts the user session container and saves the terminal information (username, URL, and state) as a new entry in Agave's Metadata API on behalf of the user. IPT leverages Docker Swarm to provide an elastic container infrastructure to deploy user session environments. User sessions are encapsulated within a single Docker container. Each container represents a unique, isolated container with persistent storage upon login. The user's session environment contains a terminal emulator, IPT software, parallel libraries, and compilers. The swarm cluster provides load balancing as well as scalability since the nodes can be added or removed on the basis of the requirement.

Starting and stopping these containers as part of user login/logout takes too long to do synchronously within a web request, so users are notified up front if their container is still starting up. Standard polling is used by the User Interface (UI) and backend server to track container status and update UI.

III. Recycling, Reuse, and ROI

In its Cyberinfrastructure Vision for 21st Century Discovery white paper, the NSF points out how critical cohesive cyberinfrastructure and software platforms are to competitiveness going forward. The IPT project is yet another data point in support of this vision. Minimally replicating the functionality of the IPT gateway "from scratch" would easily take months of dedicated development time. It is unlikely that the code would benefit from developers with strong backgrounds in security, HPC, data management, distributed systems, database and system administration, networking, devops, graphic design, web development, web standards, and real-time communication. Once built, the code would carry 100% of the technological debt associated with its development. Since we only had enough funding for N months of total development time over the life of the project, we would then instantly have to switch gears in order to develop an open source community to maintain the code. Changes in storage, resource availability, hosting, *etc.* would carry with them significant risk and threaten to derail the entire project.

In practice, none of those turned out to be issues because we leveraged the existing cyberinfrastructure to handle the majority of the gateway's complexity for us. This, in turn, allowed us to develop the initial IPT gateway with minimal effort compared to the time it would have taken had we not taken advantage of the existing infrastructure. This helped us to quickly engage our community and incorporate initial feedback, and shield our users from the impact of major hardware and network upgrades to the underlying HPC infrastructure at TACC. While the IPT gateway focuses on a niche use case, the gateway technology stack is highly reusable in part or aggregate. Abaco is actively used on several projects to handle everything from automated building of Singularity images, to event-driven Functions-as-a-Service. Agave is actively used by researchers worldwide to bridge the gap between HPC and the web. Any gateway could take these technologies independently and realize the same benefit as the IPT gateway.

Other projects with a need for remote, interactive access to HPC resources and batch execution of well-defined tasks could take the existing IPT gateway's Django code, update the form fields and style-sheets, create a Docker image containing the new interactive application, update the Agave application's IPT gateway-specific calls to run its batch jobs with the ones they would like to use, and customize the Abaco code for their resources. As future work, IPT gateway code will be refactored for reducing the coupling between some of the code components so that it can be reused by other projects with minimum coding effort.

IV. IPT Gateway User Interface

The IPT gateway frontend is built using Bootstrap 3 and Django's Jinja2 templates. A standard horizontal tabbed layout is used to organize the site. Each tab corresponds to a page in the web application. As shown in Figure 3, each page represents a unique user interaction with the software.

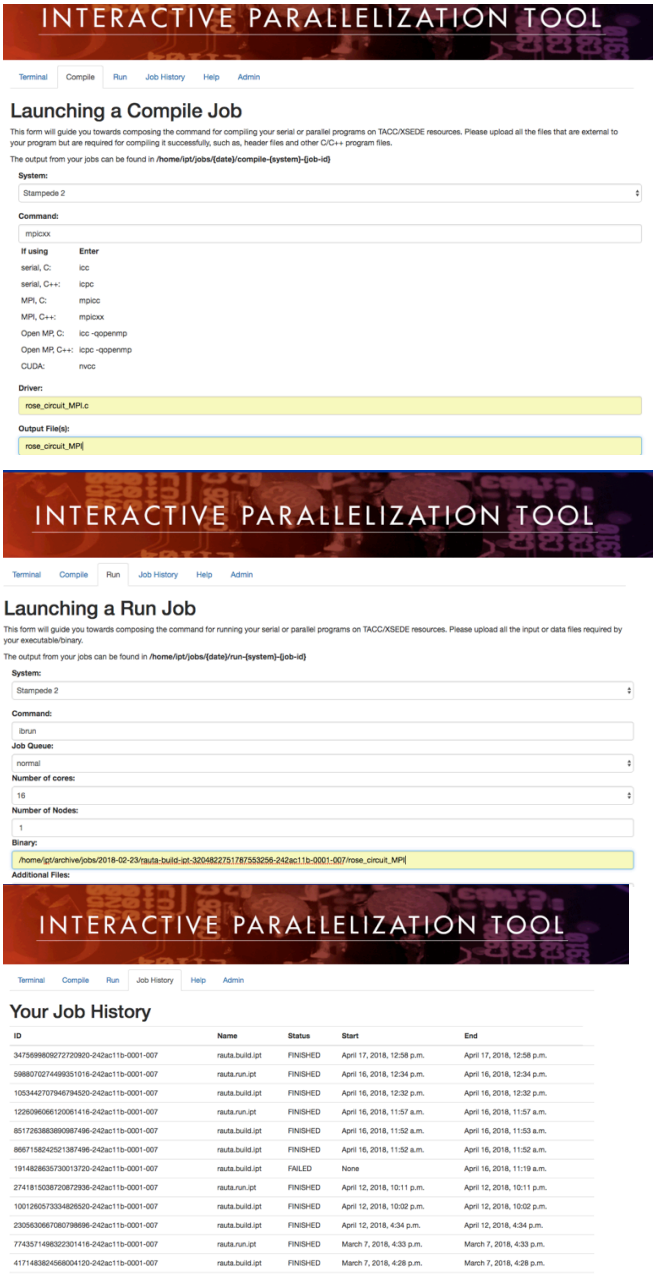

Figure 3. Screenshots of the IPT Gateway. From top to bottom: Batch compilation on remote HPC system, batch execution of HPC jobs on HPC system, batch job history

When a user first logs in, they are taken to the Terminal page where they have access to an interactive web-terminal running on their own Docker container. From there, they can interact with their code as they normally would. The Compile and Run tabs present forms that allow users to replicate their

interactive builds remotely on their target HPC systems. The History tab shows a detailed history of the user's previous build and run requests. From this page the full history, including output, logs, and parameters can be reviewed.

V. Related Work

The IPT gateway draws on numerous other open source applications and technologies common to many science gateways. The Wetty terminal [12] is a popular web-based terminal emulator. Butterfly [13], tty.js [14], and Guacamole [15] are other options, but offer hosting, networking, port, and security restrictions that are less attractive for our particular use cases. Appsoma [16], HubZero [17], Open OnDemand [18], and Jupyter [19] all offer web-based terminal access and the ability to author form-based launching of HPC applications. During the initial project planning, HubZero and Jupyter Hub were considered as all-inclusive solutions to build the gateway. Both have strong user communities, active support channels, and healthy release cycles. Further, we were familiar with the project teams from other projects. An early prototype of IPT was in fact deployed on a HubZero instance named as DiaGrid for the feasibility analysis [20].

We decided against any of the well-established gateways frameworks for the IPT gateway due to the hosting and administrative overhead required for the associated software stacks. Without funding for system administrators, dedicated hosting, or an operations team, we knew we had to leverage as much professionally hosted and managed software as possible. While we were confident there *existed* a way to scale-out HubZero in the event of unexpected growth, we had no experience doing that, nor could we commit to scaling every aspect of the infrastructure to pull it off. Jupyter gave the most flexibility with respect to the overall implementation flexibility, but also presented a challenge because we could not control the user experience with it. Given the amount of training currently done with IPT, and our stated desire to move all of that training onto the IPT Gateway, the notebook environment was a liability more than a benefit.

Various other frameworks and middleware options exist that might have also helped us achieve our goals. Eclipse Che [21], and Cloud 9 [22] provide first class web-based IDE environments with well-designed plugin systems and active communities behind them. Apache Airavata [23] and OpenStack Heat [24] all provide many of the backend platform features leveraged by our gateway, but all required significant learning curves, hosting, management, and ambiguous sustainability models for our project. In the end, we decided to build upon the Agave Platform, Abaco, and a vanilla Django web skeleton familiar to our initial developers. This stack gave us operational familiarity, minimal technological debt, and a familiar programming language.

VI. Conclusion

IPT has been deployed to the cloud to lower the adoption barriers to parallel programming and HPC. With the help of the its gateway, IPT can now be conveniently accessed through a web-browser. Not only can the users generate parallel programs through the IPT gateway, they can also test the generated programs for accuracy and performance on the supercomputers such as Stampede2, Lonestar5, and Comet. The IPT gateway is being developed iteratively and several features for improving the usability of the gateway and supporting community-building efforts will be added in future.

Acknowledgment

We are very grateful to NSF for funding IPT through award # 1642396. We are also very grateful to TACC, XSEDE (NSF award # ACI-1548562), and SGCI (NSF award # ACI-1547611) for providing resources and support for the IPT project. The deployment of IPT in the cloud is enabled by Agave (NSF awards # 1450459, 1127210) and Abaco (NSF award # 1740288) and we are grateful for these too.

References

[1] R. Arora, J. Olaya, and M. Gupta, "A Tool for Interactive Parallelization," In Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment (XSEDE '14), ACM, New York, NY, USA, Article 51, 8 pages, 2014.

[2] W. Gropp, E. Lusk, and A. Skjellum, "Using MPI: Portable Parallel Programming with the Message-Passing Interface", MIT Press, 1999.

[3] The OpenMP API specification for parallel programming: accessed on May 16, 2018: http://openmp.org/wp/

[4] What is CUDA, accessed on May 16, 2018: https://developer.nvidia.com/what-cuda

[5] K. Lawrence, M. Zentner, N. Wilkins-Diehr, J. Wernert, M. Pierce, S. Marru, and S. Michael. "Science gateways today and tomorrow: positive perspectives of nearly 5000 members of the research community," *Concurrency and Computation: Practice and Experience* 27, No. 16 (2015): 4252-4268.

[6] Agave Platform, accessed on May 16, 2018: https://agaveapi.co/

[7] Abaco, accessed on May 16, 2018: https://github.com/TACC/abaco

[8] Extended Sciences and Engineering Discovery Environment (XSEDE), accessed on May 16, 2018: https://www.xsede.org/ecosystem/resources

[9] LoneStar5 system, accessed on May 16, 2018: https://portal.tacc.utexas.edu/user-guides/lonestar5

[10] Django, accessed on May 16, 2018: https://www.djangoproject.com/

[11] Agavepy, accessed on May 16, 2018: https://github.com/TACC/agavepy

[12] Wetty, accessed on May 16, 2018: https://github.com/krishnasrinivas/wetty

[13] Butterfly, accessed on May 16, 2018: https://github.com/buttterflyframework

[14] tty.js, accessed on May 16, 2018: https://github.com/chjj/tty.js/

[15] Guacamole, accessed on May 16, 2018: https://guacamole.apache.org/

[16] AppSoma, accessed on May 16, 2018: https://github.com/appsoma

[17] Hubzero, accessed on May 16, 2018: https://hubzero.org/

[18] OpenOnDemand, accessed on May 16, 2018: https://github.com/OSC/Open-OnDemand

[19] Jupyter, accessed on May 16, 2018: http://jupyter.org/

[20] R. Arora, K. Chen, M. Gupta, S. Clark, and C. Song, "Leveraging DiaGrid hub for interactively generating and running parallel programs," In Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure (XSEDE '15), ACM, New York, NY, USA, Article 44 , 8 pages, 2015.

[21] Ecplise che, accessed on May 16, 2018: https://www.eclipse.org/che/

[22] AWS Cloud 9, accessed on May 16, 2018: https://aws.amazon.com/cloud9/

[23] Apache Airavata, accessed on May 16, 2018: https://airavata.apache.org/

[24] OpenStack Heat, accessed on May 16, 2018: https://wiki.openstack.org/wiki/Heat

[25] OpenFaas, accessed on May 16, 2018: https://github.com/openfaas/faas