Planning as a Service

Yi Ding, Cam Cunningham, Christian Muise, [1] and Nir Lipovetzky [2]

[1] Queen's University, Kingston, On, Canada
[2] The University of Melbourne, VIC, Australia
firsname.lastname @{unimelb.edu.au,queensu.ca}

Abstract

Planning as a service (PaaS) provides an extendable API to deploy planners online in local or cloud servers. The service provides a queue manager to control a set of workers, which can easily be extended with one of several planners available in PLANUTILS. PaaS is designed to overcome the limitations of the existing online solver.planning.domains interface and widen the adoption of planning technology in education, research, and industry.

Demo: https://nirlipo.github.io/project/paas/

Introduction

Since its inception in 2016, the online solver.planning.domains (Muise 2016) has been the mainstay remote planning service for planning researchers, enthusiasts, and students alike. With over five million individual solver calls, the service has remained largely unchanged in its lifetime. It houses a single solver (a version of LAPKT (Ramirez, Lipovetzky, and Muise 2015) that runs extremely quickly) and restricts the solve resources to 10 seconds and 500Mb only. Further, it operates with a single worker queue and only allows a single solve at any given time.

The seamless nature of the online solver (in concert with the other planning.domains initiatives) has proven itself to be a vital resource for settings such as education (where students repeatedly send their partial models for solving) as well as the wider adoption of planning (we are aware of several labs that use the online solver in robotic settings). However, the limitations that come with the existing service are clear and increasingly recognized as drawbacks worth addressing.

It is with this motivation that we introduce PaaS – Planning-as-a-Service. By building on the work of PLANUTILS (Muise et al. 2022), an initiative to provide turn-key terminal access to a growing array of planners and planning utilities, we developed an entirely new web framework and computing service capable of several core improvements over the previous online solver:

1. Planner configuration can be made as part of the API call.
2. Several classical planners and planners for other formalisms are now supported.
3. We can seamlessly scale the number of concurrent workers when the service is under high demand.
4. Under an opt-in process, users can contribute their model data to the initiative for research purposes (planning model data to be released to the community under a permissive license).

Similar to the previous planning service, PaaS is open-source and freely available:

https://github.com/AI-Planning/planning-as-a-service

Planning as a Service (PaaS)

PaaS consists of a 1) task queue manager to handle planning requests asynchronously in real-time, spawning new workers on multiple threads and CPUs as needed, 2) a message broker to connect the API with the queue manager, allowing to monitor the running and completed tasks, 3) an API to expose the services provided by planutils, along with their running arguments, and 4) a plugin for the editor.planning.domains (Muise and Lipovetzky 2020).

Getting Started

PaaS has been Dockerized for seamless deployment. Once the codebase is cloned, the user needs to specify a configuration file with the intended *credentials* to set up the monitoring system, the *memory* and *time* limit of each worker, and the *number of workers*. PaaS Dockers' are started by first building the latest planutils publicly available image, and then invoking the API, monitor system, and specified workers via two commands:

```
docker build -t planutils:latest .
docker-compose up -d --build
```

Two endpoints are then exposed: the API through port 5001, and the monitor service on port 5555.

Adding a Planner

Available planners are added via the Dockerfile specification of a worker. The worker uses planutils, allowing the installation of any planner available in planutils as a package. E.g., adding the Dual-BFWS planner (Lipovetzky and Geffner 2017) to the worker, can be accomplished by adding the following line:

```
RUN planutils install -f -y dual-bfws-ffparser
```

Planner Arguments and Service

We refer to (Muise et al. 2022) for more information on setting up a planner as a new package in planutils. We've extended the Manifest to expose different services and their arguments. We provide a template called *planner* to specify the `solve` endpoint service along with the domain and problem files as arguments. The service will use the command specified in `call`, and return all generated filenames matching the glob expression **plan**.

```
"args": [
      {
        "name": "domain",
        "type": "file",
        "description": "domain file"
      },
      {
        "name": "problem",
        "type": "file",
        "description": "problem file"
      }
],
"call": "{package_name} {domain} {problem}",
"return": {
  "type": "generic",
  "files": "*plan*"
}
```

We can then add a service to Dual-BFWS manifest:

```
"endpoint": {
    "services": {
      "solve": {
        "template": "planner",
        "call":"{package_name} {domain} {problem} plan"
      }
    }
```

This uses the standard planner template and overwrites the `call` property to match the run command for Dual-BFWS package.

We allow four type of arguments: `file`, `int`, `string`, and `categorical`. The latter can be used to specify a set of complex arguments similar to a select option. There are three types of return files: `generic`, when the files contain plans following the standard format used in the International Planning Competitions, i.e. one action per line, `log`, when plans are not compliant, and `json`, when plans are specified via a JSON format. This information is used by the API `adaptors`, defined for each application that intends to parse and use the returned files in a required format. Currently, we have implemented an adaptor for the online editor. New adaptors can be added and selected through our API. Otherwise, the files would be returned as text. The documentation of a service in the worker can be viewed via http://localhost:5001/docs/{package_name}.

How to access the service

The API for each service in each package will be exposed through http://localhost:5001/package/{package_name}/{package_service}. Below is a snippet using the service via Python.

```
import requests
import time
from pprint import pprint

reqBody = {
"domain":"(define (domain BLOCKS) ...)",
"problem":"(define (problem ... )"
}
ip="http://localhost:5001"
service = "/package/dual-bfws-ffparser/solve"
id=requests.post(ip + service, json=reqBody).json()
job=requests.post(ip + id['result'])

print('Computing...')
while job.json().get("status","")== 'PENDING':
    job=requests.post(ip + id['result'])
    time.sleep(0.5)

pprint(job.json())
```

The `id` variable contains the link to access the result and status of the job in the queue. We can then access the result of the job or use a custom adaptor to parse it. E.g., we can use the online editor adaptor to get the result from the queue with the following lines:

```
adaptor = {"adaptor":"planning_editor_adaptor"}
job=requests.get(ip + id['result'], json=adaptor)
```

The PaaS service has already been deployed and integrated into the online editor.planning.domains as a plugin https://github.com/AI-Planning/planning-as-a-service-plugin, allowing for any planner exposing its services via planutils to be used within the editor.

Use Cases

The core functionality of PaaS has been incorporated in the online editor, enabling students to easily solve different subsets of PDDL, as well as testing the capabilities of the diverse suite of techniques implemented in existing planners. To further ease the adoption of planning, preliminary support also exists for using PaaS in VScode via the existing PDDL extension, as well as in Planimation (Chen et al. 2020), which would currently allow for plans from different classical planners to be easily inspected, and in the future to visualize plans using fragments of PDDL beyond classical planning. Finally, PaaS can easily be deployed in a local/-cloud server to provide planning services to other applications, such as robotic platforms.

Acknowledgments This work has been partially funded by AIJ to promote AI Research "Enabling Education of AI Planning"

References

Chen, G.; Ding, Y.; Edwards, H.; Chau, C. H.; Hou, S.; Johnson, G.; Sharukh Syed, M.; Tang, H.; Wu, Y.; Yan, Y.; Gil, T.; and Nir, L. 2020. Planimation. *arXiv preprint arXiv:2008.04600*.

Lipovetzky, N.; and Geffner, H. 2017. Best-First Width Search: Exploration and Exploitation in Classical Planning. In *Proc. AAAI*.

Muise, C. 2016. Planning. domains. *ICAPS system demonstration*.

Muise, C.; and Lipovetzky, N. 2020. KEPS Book: Planning. Domains. *Knowledge Engineering Tools and Techniques for AI Planning*, 91–105.

Muise, C.; Pommerening, F.; Seipp, J.; and Katz, M. 2022. PLANUTILS: Bringing Planning to the Masses. In *ICAPS: System Demonstrations*.

Ramirez, M.; Lipovetzky, N.; and Muise, C. 2015. Lightweight Automated Planning ToolKiT. http://lapkt.org/.