1.1

Employing Task Parallelism to Facilitate Dynamic Comparison of Model Output

Hannah Aizenman [*] and Michael Grossberg

Glasslab, City College of New York, New York, New York

David Brown and Mary Haley

CISL, NCAR, Boulder

ABSTRACT

The analysis of many climate models, such as the Community Earth System Model (CESM), require extensive and expensive computing and storage resources. Since these resources are often out of the reach of many young scientists and small research groups, the goal of this National Consortium of Atmospheric Research's (NCAR) Computational Information Systems Laboratory's (CISL) Summer Internships in Parallel Computational Science(SIParCS) project was to make a commonly used climate model diagnostic tool available through the web so that users could make use of NCAR resources in a simple and public way. The user facing interface is written in HTML/JavaScript and is built on top of a RESTful API implemented using the Pyramid web framework. An advantage of the Pyramid web framework is its lightweight plug and play architecture which makes it easier to later maintain, customize, and extend the tool as the needs of the user and application grow. The gevent library was used to add in asynchronous task execution so that each execution of the diagnostics could be independent. This will enable web-based real-time monitoring of the status of a diagnostic run. The results of the run are treated as a RESTful resource so that they can be obtained as a compressed archive or file list, for example, using the same URL, and to make it simpler to later integrate a database or some caching scheme. The UI is separated from the server side API to give developers the ability to easily add in new interfaces without destroying functional ones.

A major aspect of building the web interface was rewriting the CESM Ocean Model Working Group's (OMWG) model diagnostics c-shell driver scripts as a Python library. This was done mainly to simplify the process of running the diagnostics with user defined settings, but also to improve their usability, maintainability, robustness, and extensibility. Python was chosen in large part because it has extensive support for managing calls to shell utilities, which was crucial because the diagnostics depend very heavily on numerous shell and NCL scripts that would take an extensive amount of time to convert to Python. The library wraps the Swift parallel scripting language version of the diagnostics so that the NCL scripts at the core of the diagnostics can be run in parallel instead of serially. Functions for creating swift configurations were also built into the library to make working with swift easier. The web version takes full advantage of this by running independently configured jobs on every call and returning the results as either an archive, a styled folder, or a plain folder, depending on the URL. This flexibility was a major design goal in creating this tool, because that makes it easier for users to fit it into their work-flow, which is key to any form of widespread adoption of the tool. We created a maintainable and highly documented web interface to the parallel version of CESM-OMWG diagnostics so that researchers outside of NCAR could more easily work with the CESM and so that researchers could extend this tool to other datasets and diagnostics, or build their own, thereby increasing public access to both data and the tools used to understand it.

1. Introduction

The Community Earth System Model (CESM) generates terabytes of data on a single run, creating undue hardship for many young scientists and small research groups who would want to work with it, so one of the projects for this year's Summer Internships in Parallel Computational Science(SIParCS), which is run by the Computational Information Systems Laboratory's (CISL) at NCAR, was to make the analysis tools for the ocean component of the CESM more accessible. Before work began on the web interface, the control scripts for the Ocean Model Working Group's (OMWG) Parallel Ocean Program (POP) diagnostics were rewritten to work within a web-framework by turining them into a Python library. Python was used because it has extensive native support for running external scripts, which make up a large part of the diagnostics. A web-interface (WebDiag) was also created so that researchers could explore the diagnostics more easily. The interface is designed as a very thin wrapper on top of the PyDiag library and the user facing interface is written in HTML/JavaScript and is built on top of a RESTful API implemented using the Pyramid web framework. Throughout the development, a modular architecture was stressed so that the tool could be adapted as requirements change.

2. Prior Work

The project is a direct successor to the SWIFT OMWG POP DiagnosticsCESM Ocean Model Working Group (2012) developed by the CISL and Climate and Global Dynamics (CGD) at NCAR and the Parviz group at Argonne National Lab (ANL). The diagnostics analyze results of the CESM Ocean model given a model run as input. There are three main diagnostics scripts: seasonal and time-mean diagnostics of a single model run (popdiag), time-series diagnostics for one rune (popdiagts), and the difference between two model runs or two time-periods of the same run (popdiagdiff). These scripts, written in c-shell, setup the environment variables for a run and then either call the NCL scripts directly or call a swift script that calls the NCL scripts in parallel. The NCL scripts generate the tables and figures that make up the body of the analysis, while other c-shell, awk, and Perl scripts do miscellaneous processing. The diagnostics yield variables derived from the models, for example salt content and ocean temperature. The c-shell scripts also can also create a webpage by copying the results into a folder, but it depends on the output file names always being known and only displays some results.

3. Software Architecture

WebDiag uses a Service Oriented Architecture (SOA), as seen in figure 1 where the UI, tools for getting the data and diagnostics, and the databases and PyDiag are all independent. An SOA approach is used to allow for greater flexibility in deployment due to the computational and storage requirements of the analysis and the security requirements at NCAR, because in deployment the computational side can be deployed on a cluster hidden behind firewalls while the UI can be deployed anywhere.

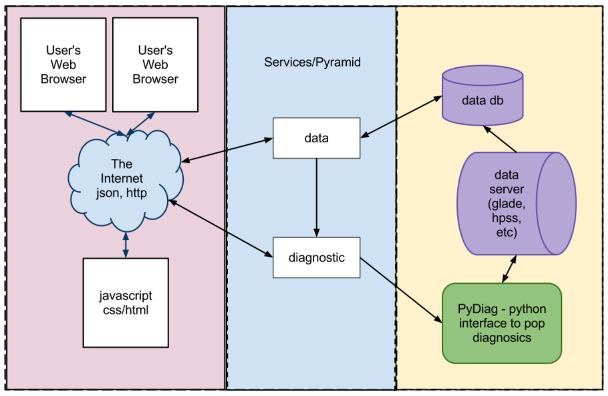

FIG. 1. The figure illustrates the three distinct sections of the tool: the diagnostics handled by the standalone PyDiag library, the web framework layer that translates user input into PyDiag function calls, and the JavaScript/HTML UI layer that sits in the users browser.

a. *PyDiag*

PyDiag is the library version of the POP diagnostics. The guiding principal was to simplify, so the dependencies were isolated and reducded and the various methods for setting user configurations (environment variables, XML files, tab delimited, cdm, etc.) were all changed to one format (dictionaries) and utilities are used to generate the correct types. A secondary principal was ease of use, so PyDiag is compatible with Python¿=2.4 as older versions of Python are common on research machines. The modular architecture also makes it easier to add in new diagnostics, and the extensive documentation and tests provide templates for how to do so

b. *WebDiag Interface*

WebDiag's web service functionality is provided using the Pyramid Python Web Framework (Pyramid Development Team (2010)). Pyramid was chosen because it is can be used to write fairly lightweight applications. As seen in 2, the web-service layer takes the users request, validates it, and then passes it down into an interface that grabs the information about the data from the database and sends all that information to PyDiag so that it can do the computation. A database is used to store information about the model and observational data because diagnostic runs tend to use the same datasets. To simplify debugging web errors, the URL logic (validation and parsing) is kept seperate from the code that manages the run.

c. *User Interface*

As shown in figure 3 and figure 4, using a RESTful interface means that the website is modeled as a collection of

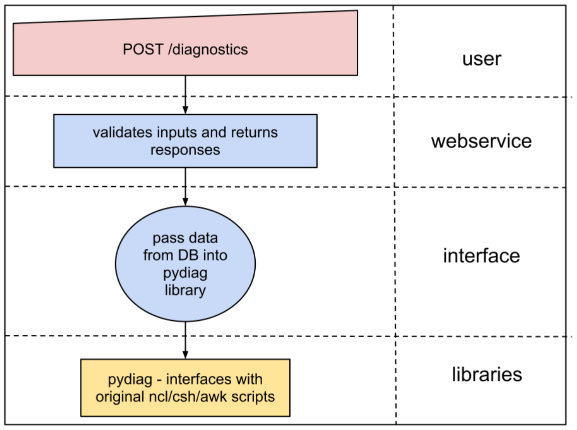
POST /diagnostics
user
validates inputs and returns responses
webservice
pass data from DB into pydiag library
interface
pydiag - interfaces with original ncl/csh/awk scripts
libraries

FIG. 2. The user makes a selection that gets encoded in a URL, which gets parsed by a web-service. The contents of the url are then used to retrieve data from the database. The data and any other necessary arguments in the URL are then passed into PyDiag, which runs the diagnostics.

resources. In this application, the resources are the original dataset, the available list of diagnostics, and the result. This allows the user to write a dynamic interface based on the resources, as in 3, where the menu is generated based on the chosen option. In figure 4, the diagnostic is started, thereby creating the results resource, using **POST /diagnostic**, which yields a unique identifier for the results once it finishes. The resource is then retrieved using is **GET /diagnostic/{d}**, with optional keywords to specify the format of the results. Pyramid simplified the process of using RESTful architecture because it has really good support for modeling every resource as an object.

4. Conclusion

We created a web based and API interface to the OMWG POP diagnostics to simplify and somewhat automate running the diagnostics. Our multiple interfaces using a variety of modular architecture gives the user the power to interact with the diagnostics in a number of ways, or even ignore the diagnostics entirely if the user just wants the data. By keeping the components modular, we facilitate reuse of the code in new applications. The architecture is designed to encourage future work in expanding to other diagnostics and managing the very large results the diagnostics yield because those are just more plug-ins.

Acknowledgments.

This work was done as a summer project as part of SIParCS at NCAR-CISL, with mentoring by David Brown and Mary Haley (NCAR-CISL). The following people also provided invaluable feedback: Susan Bates and Gokhan Danabasoglu (NCAR-CGD), Sheri Mickelson (ANL), Paul Goodman, Nathan Wilhelmi, Wei Huang and Rick Brownrigg(NCAR/CISL) and Michael Grossberg and Jeremy Neiman (Glasslab, CCNY) .

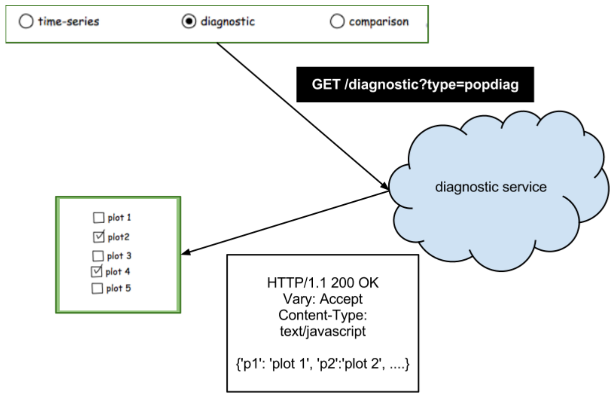

FIG. 3. The GUI menu makes a request for every type of analysis provided by popdiag, and since the diagnostics are a service, the server response with a list of plots. This list is then used to populate a secondary menu.

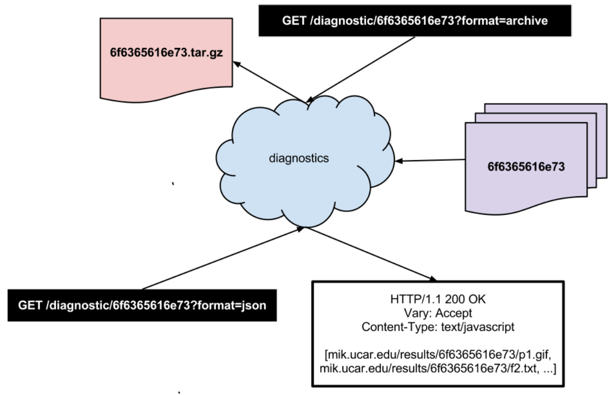
GET /diagnostic/6f6365616e73?format=archive
6f6365616e73.tar.gz
diagnostics
6f6365616e73
GET /diagnostic/6f6365616e73?format=json
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript
[mik.ucar.edu/results/6f6365616e73/p1.gif, mik.ucar.edu/results/6f6365616e73/f2.txt, ...]

FIG. 4. Multiple ways of viewing the same results by retrieving a resource and then applying a view to it.

REFERENCES

CESM Ocean Model Working Group, 2012: CESM1.0: Parallel Ocean Program (POP2). `http://www.cesm.ucar.edu/models/cesm1.0/pop2/`.

Pyramid Development Team, 2010: The pyramid web application development framework. `http://www.pylonsproject.org/`.