1 Introduction

The Asteroseismic Modeling Portal (AMP) provides a web-based interface for astronomers to derive the properties of Sun-like stars from observations of their pulsation frequencies. The primary motivation for AMP is NASA’s recent launch of the Kepler satellite and its mission to identify potentially habitable Earth-like planets. Kepler detects planets by observing extrasolar transits – brief dips in observed brightness as a planet passes between its star and the satellite – that identify the size of the planet relative to the size of the star. Asteroseismology can be used to determine the precise absolute size of the star and thus the size of the planet. AMP makes an asteroseismology model available to a broad international community of researchers, facilitating automatic model execution and simplifying data sharing among research groups, consequently helping produce a uniform analysis of asteroseismic data for stars of interest.

AMP has been designed since its inception as a TeraGrid science gateway. Many of the best practices and procedures for developing and deploying science gateways on the TeraGrid were proposed coincident with our initial exploration of targeting TeraGrid as AMP’s computational platform [8]. As such, AMP serves as an example of constructing a new science gateway specifically for TeraGrid cyberinfrastructure rather than the common case of extending an existing gateway to utilize TeraGrid.

The straightforward workflow implemented by AMP also provided an opportunity to develop a new science gateway while exploring a new architecture, web application framework, and supporting technologies. Many of the prior science gateway projects at the National Center for Atmospheric Research (NCAR) followed the general design pattern typical of many gateways today by using Java to implement service-oriented architectures (SOAs) and web portals. Most notably divergent from our prior work, AMP does not use an application-specific service-oriented architecture and is not written in Java. Instead, we utilized only Grid middleware components common to all TeraGrid resource providers and implemented the solution in Python using the Django framework. AMP’s architecture separates the Web-based user interface and the workflow system performing Grid operations, isolating interactive users from TeraGrid operations. The AMP architecture has been constructed with the goal of facilitating easy deployment on current TeraGrid-managed resources without any resource provider assistance.

2 Background

The asteroseismology software executed by AMP consists of two components: a forward stellar model and a genetic algorithm (GA) that invokes the forward model as a subroutine. The forward stellar model is the Aarhus Stellar Evolution Code (ASTEC) [3], a single-processor code that takes as input five floating-point physical parameters (mass, metallicity, helium mass fraction, and convective efficiency) and constructs a model of the star’s evolution through a specified age. The output of the model includes observable data about such as the star’s temperature, luminosity, and pulsation frequencies.

In order to derive the properties of distant stars from observations, ASTEC is coupled with the MPIKAIA parallel GA [4]. The GA creates a population of candidate stars with a variety of physical parameters, models each star using ASTEC, and then evaluates each candidate star for similarity to the observed data. Over many iterations, the GA converges to identify an optimal candidate star that has the properties most likely to produce the observed data.

AMP supports two modes of execution from its web-based user interface: running the forward model with specific model parameters (a “direct model run”), and executing the GA to identify model parameters that produce

Figure 1: AMP stellar model optimization workflow.
observed data (an "optimization run"). The optimization run workflow consists of an ensemble of independent GA runs and many computational tasks (see Figure 1). For each optimization run, multiple separate GA runs are executed and allowed to converge independently in order to provide confidence in the optimality of the final result. Due to system walltime limitations, each GA run may require several invocations of the executable to propagate the GA to completion. When the ensemble is complete, the best solution is evaluated using the forward model to produce detailed output for presentation and analysis.

In the current configuration, each optimization run consists of four GA runs executed in parallel, and each GA propagates a population of 127 stars (using 128 processors) for 200 iterations. However, as the population converges, the iteration time decreases, so the 200 iterations can be performed in about 160x of the first iteration’s time. The stellar model has been benchmarked on four TeraGrid platforms (see Table 1). When considering SU charging factors and the model performance, the TACC systems are most efficient platforms for this model.

### Table 1: Measured stellar model benchmark run time and estimated optimization run time and SU charge for selected TeraGrid systems.

<table>
<thead>
<tr>
<th>System</th>
<th>Stellar Model Run Time (min)</th>
<th>Optimization Run (Genetic Algorithm) Run Time (h)</th>
<th>CPUh</th>
<th>SUs/CPUh</th>
<th>TeraGrid SUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>NCAR Frost</td>
<td>110.0</td>
<td>293.3</td>
<td>150,187</td>
<td>0.558</td>
<td>83,804</td>
</tr>
<tr>
<td>NICS Kraken</td>
<td>23.6</td>
<td>61.9</td>
<td>31,723</td>
<td>1.623</td>
<td>51,486</td>
</tr>
<tr>
<td>TACC Lonestar</td>
<td>15.1</td>
<td>40.4</td>
<td>20,670</td>
<td>1.935</td>
<td>39,996</td>
</tr>
<tr>
<td>TACC Ranger</td>
<td>21.1</td>
<td>56.2</td>
<td>28,771</td>
<td>1.644</td>
<td>47,229</td>
</tr>
</tbody>
</table>

Table 1: Measured stellar model benchmark run time and estimated optimization run time and SU charge for selected TeraGrid systems. An optimization run performs 200 GA iterations and requires about 160x the model benchmark time to complete. For computational time, each GA executes four 128-processor jobs.

3 Design

AMP has been designed as a TeraGrid science gateway and its architecture and implementation are intended to simplify its utilization of TeraGrid computational resources. Not only does AMP implement the emerging TeraGrid features supporting end-to-end user identification and accounting, but the architecture has been designed to allow only very carefully mediated access to TeraGrid resources using AMP’s community account credential.

3.1 Separation of Users and Grid Operations

One concern often associated with science gateways is their use of a shared credential to submit jobs on behalf of a community of individual gateway users. Gateways are required to maintain user registries and the ability to associate a Grid request with a gateway user, and the new GridShib SAML extensions extend the disambiguation ability to resource providers themselves [5]. However, an underlying risk remains: a science gateway typically runs a publicly accessible web server and also must possess the credentials necessary to access many machines on the TeraGrid.

Our primary response to this concern is directly reflected in the AMP architecture that separates the users from the community account credential by placing them on distinct servers (see Figure 2). The user interacts with a web portal located on one publicly-accessible server, while all back-end processing and remote Grid operations are performed by a separate daemon (referred to as the GridAMP daemon) on another server. All communication between the AMP portal and the GridAMP daemon are asynchronously managed by manipulating a database located on yet another server.

The roles and privileges of the public web portal and GridAMP daemon are strictly managed and controlled. The public web portal is essentially a database-driven web server without any Grid connectivity or Grid software. The server hosting the GridAMP daemon is accessible only to the developers using SSH keys, and only GridFTP is
externally exposed to facilitate data staging via the community account credential. All input data from users is marshaled through the SQL database – a feature made possible by the extremely small (2KB) input files required by the model for each optimization run. Input files are parsed by the web server and uploaded to SQL tables with strict data type constraints. When required, the input files are regenerated from the data by the GridAMP daemon and then staged to TeraGrid systems. Moreover, SQL permissions are set so that connections from the AMP and GridAMP servers have the minimal set of database privileges required for operation.

This architecture reduces the possibility for users to execute outside the bounds of the permitted job definition construction. AMP users do not submit jobs or transfer files to TeraGrid resources directly; rather, users enter model input parameters and the GridAMP daemon creates the request and submits it to the TeraGrid on the user’s behalf.

3.2 Targeting TeraGrid CTSS Remote Services

To simplify the deployment of the AMP model on TeraGrid systems, we constructed the execution workflow to utilize only basic components provided by the Coordinated TeraGrid Software and Services (CTSS) software stack. [7]. Rather than deploying a SOA with services that encapsulate the models as we have done in the past for other projects, the GridAMP daemon directly formulates and submits GRAM execution requests and GridFTP file transfers. Thus, the model can be deployed on a TeraGrid resource as soon as the community account has been authorized and no special resource provider dispensations (e.g., custom Globus containers or separate service hosting platforms) are required.

4 Implementation Experiences

The AMP gateway and GridAMP daemon are implemented in Python 2.4 using the Django development framework [2]. Django’s primary intended use is as a web development platform, but over two software engineering iterations, we adopted Django as the underlying framework for both the AMP website and the GridAMP daemon.

In addition to its web server-related features, Django implements a full-featured object-relational model (ORM) to provide database access functionality. Given our requirement of directly controlling database table structure and access privileges, we were initially skeptical of using Django’s ORM; however, the ORM proved sufficiently flexible and powerful for use with both the website and processing daemon. Moreover, by selecting a single development framework for both components, the entire project uses a single code base to define and manipulate shared data structures across multiple servers.

The Django framework has also provided an efficient development process and supports production system administration. Django’s built-in development server allows developers to test any component of the AMP portal from a workstation without installing or configuring a standalone web server. In addition, the Django administrative interface allows easy manipulation of all underlying database tables without custom software development. Routine gateway resource and user management is performed using this built-in administrative interface from trusted internal systems.

5 Conclusions

AMP has provided an opportunity to develop a new science gateway exclusively utilizing TeraGrid computational resources. The gateway is currently available for friendly user testing, and we anticipate the first extensive use of the system to perform new asteroseismology science in mid-2009. In the future, we plan to examine possible applications of AMP’s architecture and underlying technology choices to other NCAR science gateway projects.

Acknowledgments

Thanks to Nancy Wilkins-Diehr for assistance turning AMP into a TeraGrid Science Gateway, Stu Martin for assistance with Globus GRAM auditing, and Tom Scavo for assistance with GridShib. Thanks to Margaret Murray for assisting us as we started using TACC resources, and to Victor Hazlewood and Rick Mohr for similar assistance with NICS resources. Michael Oberg prepared the NCAR TeraGrid Service Hosting Platform used to host AMP and GridAMP. Will Baird developed many prototype AMP components and features including AMP’s utilization of the SIMBAD [6] astronomical database.
Funding to integrate AMP with TeraGrid resources was provided by the TeraGrid Advanced Support Program. Computational time at NCAR was provided by NSF MRI Grant #CNS-0421498, NSF MRI Grant #CNS-0420873, NSF MRI Grant #CNS-0420985, NSF sponsorship of the National Center for Atmospheric Research, the University of Colorado, and a grant from the IBM Shared University Research (SUR) program.

References


