Parallel Tools GUI Framework

DOE SBIR Phase I Final Technical Report

Award Number: DE-SC0009499TDD

Topic/Subtopic: 2b

Principal Investigator: James Galarowicz

Company: Argo Navis Technologies LLC
PO Box 6732
Annapolis, MD 21401

Report Date: December 5th, 2013

Contents

1 Introduction

Many parallel performance, profiling, and debugging tools require a graphical way of displaying the very large datasets typically gathered from high performance computing (HPC) applications. Most tool projects create their graphical user interfaces (GUI) from scratch, many times spending their project resources on simply redeveloping commonly used infrastructure. Our goal was to create a multi-platform GUI framework, based on Nokia/Digia's popular Qt libraries, which will specifically address the needs of these parallel tools.

The Parallel Tools GUI Framework (PTGF) uses a plugin architecture facilitating rapid GUI development and reduced development costs for new and existing tool projects by allowing the reuse of many common GUI elements, called "widgets." Widgets created include, 2D data visualizations, a source code viewer with syntax highlighting, and integrated help and welcome screens. Application programming interface (API) design was focused on minimizing the time to getting a functional tool working.

Having a standard, unified, and user-friendly interface which operates on multiple platforms will benefit HPC application developers by reducing training time and allowing users to move between tools rapidly during a single session.

However, Argo Navis Technologies LLC will not be submitting a DOE SBIR Phase II proposal and commercialization plan for the PTGF project. Our preliminary estimates for gross income over the next several years was based upon initial customer interest and income generated by similar projects. Unfortunately, as we further assessed the market during Phase I, we grew to realize that there was not enough demand to warrant such a large investment. While we do find that the project is worth our continued investment of time and money, we do not think it worthy of the DOE's investment at this time. We are grateful that the DOE has afforded us the opportunity to make this assessment, and come to this conclusion.

2 Comparison of Objectives to Accomplishments

Throughout Phase I of this project we found our proposed goals to be reasonably aligned with what we have been able to accomplish.

2.1 Proposed Technical Objectives

The following technical objectives were proposed in our Phase I proposal:

1. Facilitate the rapid development of cross-platform user interfaces for new and existing parallel tools.

2. Target a stable version of Qt4 which is currently available on many existing cluster platforms throughout the DOE computing complex. This version will not be the latest available, but will allow for easier adoption on existing platforms. The framework shall also be forward-compatible with all following versions of Qt4.

3. Provide abstracted visualizations for easy inclusion in multiple parallel tools. These abstracted visualizations will accept a simple dataset. The visualization plugins will also act as dynamic libraries, which can be easily extended by tool developers looking to specialize a particular view.

4. Provide a scalable design/model which will allow tools with very large datasets to be used effectively within the PTGF.

5. Provide a standardized interface such that users will find enough similarities between tools to make learning additional ones easier.

6. Provide facilities for user learning of a new parallel tool from within PTGF, and the ability to link to online resources.

7. Assess the viability, and usefulness to DOE facilities, of providing a mechanism for network updates of PTGF plugins.

2.2 Actual Technical Objective Accomplishments

To meet objective one, we took measures to facilitate rapid parallel tool GUI development through clean and simple API design. This helps the tool developer to learn the PTGF quickly, and use it effectively with a minimum amount of code. We feel that objective one has been fully met, and was proved through the rapid development and deployment of user interfaces for two parallel tools created by the Los Alamos National Laboratory's (LANL) High Performance Computing (HPC) system support team.

We have not only met, but exceeded expectations set forth in objective two. Red Hat Enterprise Linux (RHEL) and it's open source derivative, CentOS, are a stable OS distribution used throughout the DOE computing cluster systems, which has one of the oldest stable versions of Qt that we could find in wide use, 4.6.2 released in February of 2010[1]. The PTGF seamlessly supports versions of Qt4 starting at 4.6.2 through the latest releases of Qt5 (5.1.1 at the time of this report).

Objective three was met using the existing abstract data model types defined in Qt[2], which allows tools to simply populate an object in memory with their data, and pass that model to a pool of run-time loaded view plugins. The Open|SpeedShop[4] GUI (Figure 1) currently being developed using the PTGF is subclassing one of the visualization plugins for specialization to specifically handle specialized call stack trace data in an intuitive manner. Further, we are able to demonstrate the scalability of the PTGF through the display of Open|SpeedShop's large data sets, fully satisfying objective four.

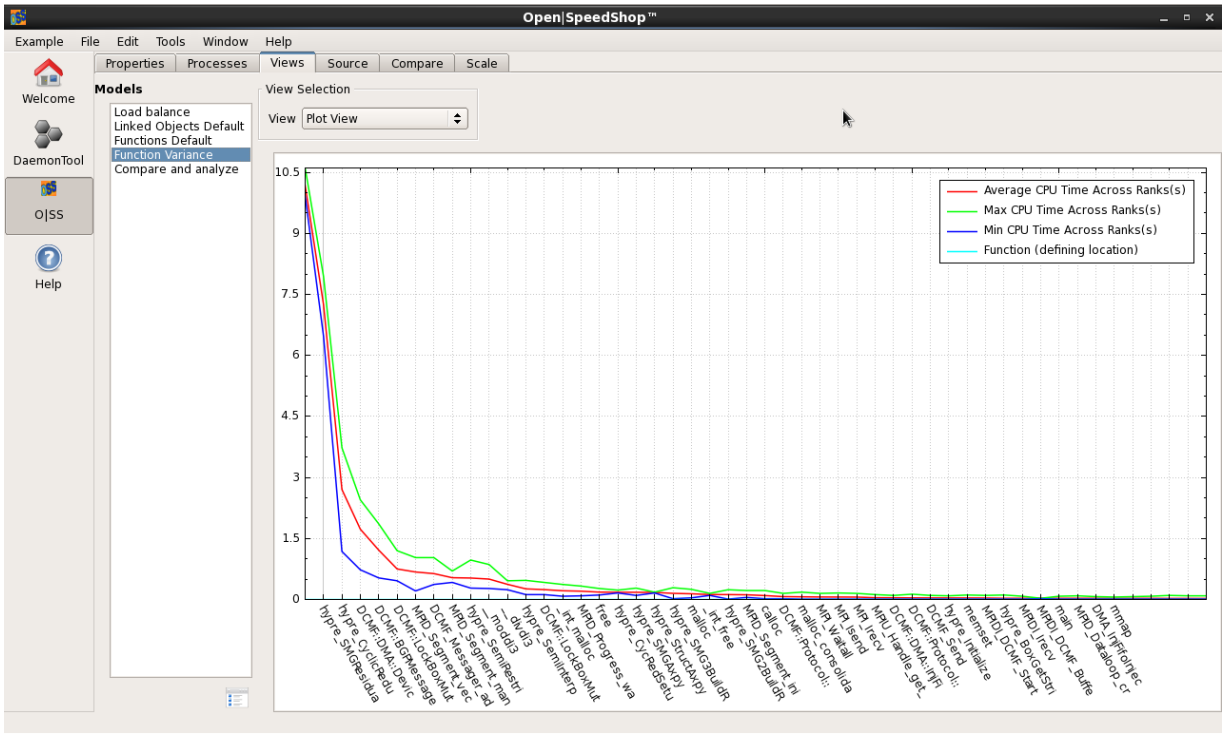

Figure 1: Demonstration of Open|SpeedShop's utilization of default PTGF visualizations, running alongside another set of parallel tools under the CBTF DaemonTool tab.

With the development of PTGF interface plugins for the two LANL tools previously mentioned, and Open|SpeedShop, we are able to demonstrate (Figure 1) that objective five has been met. The PTGF allows rapid context switching between the two sets of tools, and offers interfaces for both that appears seamless and intuitive to a user.

Objective six was met through the use of integrated help file access that can be registered by a tool's plugin, which allows the user to view compiled and uncompiled HTML-based help files from within the tool GUI. Further, links to online help resources, like YouTube demonstration videos, or tutorials on a company website, are displayed to the user on the Welcome screen (Figure 2). While the base functionality for some of these features already existed going into Phase I, the API has been cleaned up for easier development, feature "stubs" have been extended for full support, and the system is now backed by a regression testing harness ensuring quality of future additions.

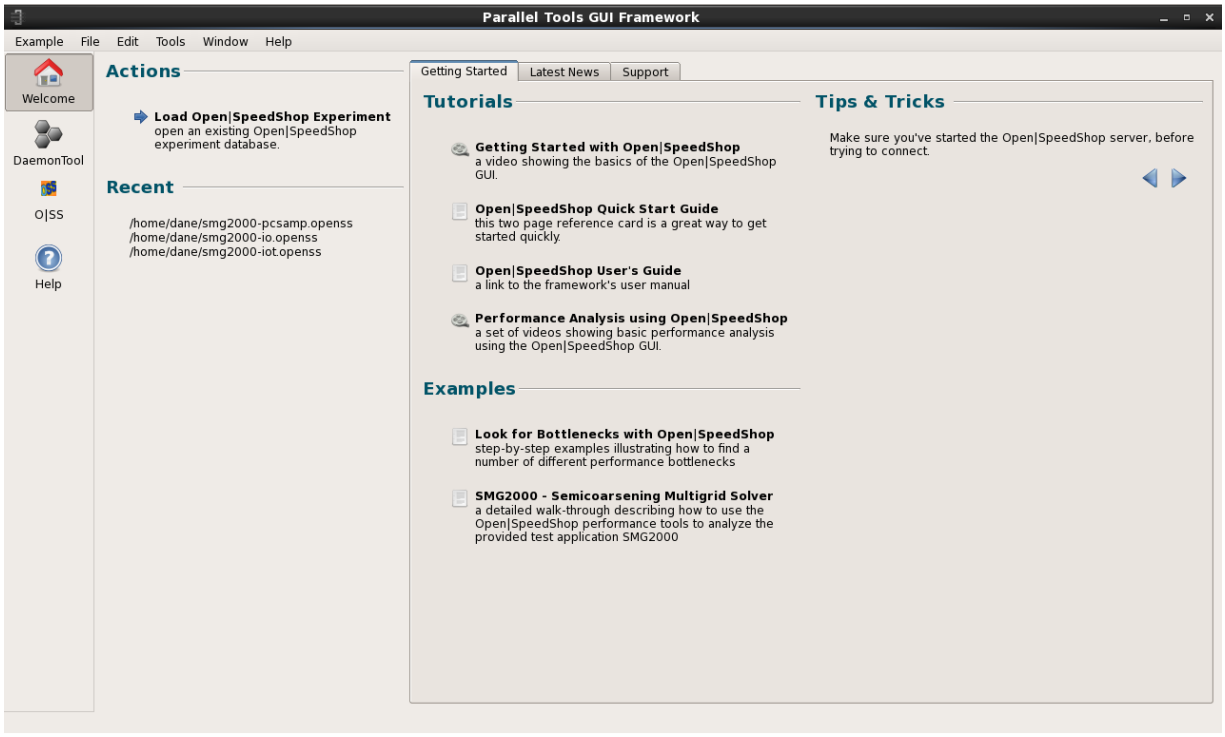

Figure 2: Integrated links to help and other documentation that can be used by tool plugins.

To meet objective seven we had several discussions with HPC support personnel at DOE Laboratories. Unfortunately, the conclusion that we came to was that while technically feasible, and a desirable feature, this was not something that could be easily implemented with existing security protocols.

3 Project Activities

With these objectives in mind, we proposed the following milestones for the Phase I work plan:

1. Core Executable and Main Window Library
2. Plugin, Action and Setting Managers
3. Extended Widget Library
4. Help and Welcome Screen Plugins
5. Abstracted Visualizations
6. Network Update Plugin

3.1 Core Executable and Main Window Library

A plugin architecture was chosen to allow components to be loaded at runtime. This empowers the cluster system administrator, or even the user, to pick and choose which tools are available within the framework, without having to recompile every part of the project.

The framework is split into three major sections, each with many components. Figure 3 illustrates this layout. The core executable (a.k.a. main executable) is a very small piece of code that is solely responsible for loading and initializing the various linked libraries.

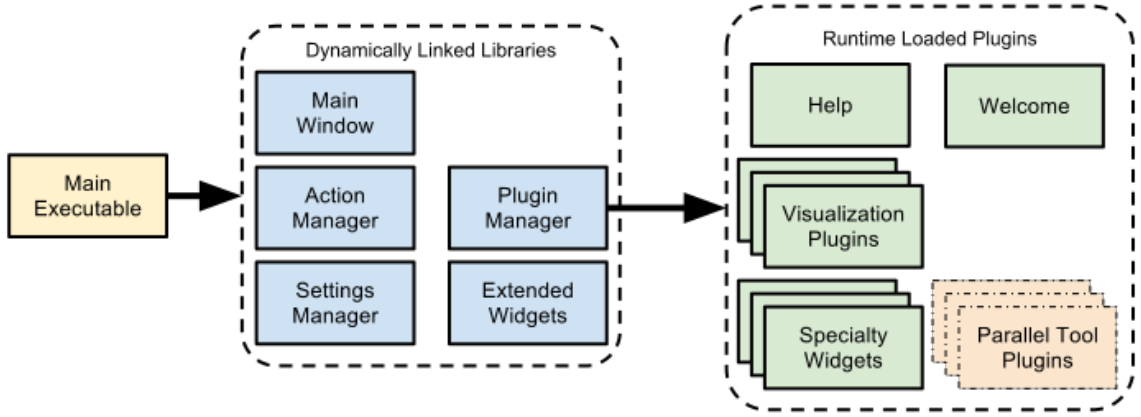

Figure 3: Logical layout of libraries and plugin architecture

These dynamically linked libraries include the main window widget responsible for presenting each of the loaded plugins registered as a main widget. Please see Figure 2 for a screenshot of the existing system with two parallel tool's registered main widgets, Open|SpeedShop and CBTF DaemonTool.

We proposed the following for Phase I: The main window manager will allow tool plugins to register a “main window” for the tool, which will act as the representation of a gateway to the user for the tool. The main window will interact closely with the notification and logging manager, the library responsible for user notifications and simplified error logging. User notifications will take the form of a notification bar at the top of the screen, a modal dialog box, or a progress bar informing the user of the status of a long running operation.

All of these items were completed by the end of Phase I. In addition, we were able extend the existing Qt debug interfaces to include an error and debug information window that can be hidden or shown by the user as needed. Further, this functionality allows a minimum notification “debug level” to be set, whereby a user will not be notified of unimportant messages using the notification bar, further reducing the message load on the user.

3.2 Plugin, Action and Setting Managers

To better facilitate rapid development of a user interface for a given tool, we planned several managers to automatically deal with specific tasks by managing objects that implement a pre-defined interface.

3.2.1 Plugin Manager

The plugin manager does what its name implies, it scans predefined and user-defined directories for plugin files that can be loaded. Plugins are required to follow a predefined interface, which allows the plugin manager to query the plugin for a list of dependencies. The manager will then sort the ordering of the plugin initializations, such that plugins that depend on other plugins to be initialized first, are.

In Phase I we have been able to, as proposed, update the plugin manager to work with the latest Qt5 architecture, as well as extend the functionality to include better plugin path searching, allowing multiple plugin paths, enabling the user to set the paths from the GUI or through environment variables. We’ve also added better handling of failures during plugin loading, and meaningful error messages empowers the user to solve problems with minimal effort.

3.2.2 Action Manager

The action manager is responsible for menu and toolbar items displayed to the user throughout the lifetime of the application. With multiple parallel tools existing within the same application, methods for preventing collisions of shortcut keys, and menu item placement is an absolute must. Prior to Phase I some initial design work had been performed on the action manager, but it had not been implemented. Having created several tools that interact with this framework, we have discovered that this is an absolute must to ensure easy creation and integration of future tools.

During Phase I we fully implemented an action manager that can handle menu creation using simple “path” descriptors to locate each clickable menu item, with priority numbering to deal with ordering. Contexts allow the menu items to be easily disabled or hidden when appropriate. These features enable the tool developer to create a quick menu system that integrates with the user interface, while having no knowledge of how other tools are using the menu system.

3.2.3 Setting Manager

Qt4 has facilities for storing and retrieving settings, however these are simplified and require that they are persisted in a flat text file in the user’s home directory. In our proposal we planned to expand functionality to allow for not just user customization, but system specific settings that can be specified by the system administrator. As well as persistence of settings using a SQL database, which will enable a central server to host the settings across multiple machines where home directories aren’t shared.

During our Phase I work, we were able to perform the above tasks with the exception of SQL database usage, due to technical limitations of the Qt framework. We were, however, able to extend the settings manager to include persisting to XML files, which can be useful in some situations.

3.3 Extended Widget Library

Many of the Qt standard widgets have more features than are necessary for a parallel tool’s GUI. However, there are several commonly needed widgets that are lacking. Further, there are many features in newer versions of Qt that are unavailable on some DOE machines due to upgrade issues. A library of extended widgets would alleviate both of these issues.

Proposed widgets included a tab widget which hides the tab bar when there is only one tab, for a cleaner initial interface; a collapsible group box for hiding, but making easily accessible, advanced functionality from novice users; placeholder text in line edit boxes for better user prompting; and a set of control widgets standard to debugging and profiling tools connecting to live applications (play, pause, step, stop, etc.).

In addition to the proposed Phase I work, we also extended the tab widget with the use of a stylesheet to improve appearance of an empty set of tabs, which is common case when the application is first opened. Further we were able to animate the collapsible group box, giving some professional flair to tool developer’s GUIs, something that users are expecting more and more with exposure to popular cell phone app interfaces.

As part of the Phase I work, we also found the need for an unproposed text console widget, that provides an easy interface for displaying textual information to the user. This can be useful for in-GUI debugging information, or CLI output from the original tool.

3.4 Help and Welcome Screen Plugins

Existing plugins for the help system and welcome screens are loaded at runtime by the plugin manager. Demonstrated in Figure 2, is a screenshot containing the welcome widget's display immediately presented to the user. The welcome plugin allows tools to register various items to help the novice user get started. Documentation, tutorials, and online videos can all be presented to the user. Prior to Phase I, only feature stubs which emulated this functionality existed. As proposed for Phase I, we implemented methods for the tool developer to register actions to the welcome screen, as well as tutorial documentation and videos, "tips and tricks" texts, and RSS feeds for news.

We also proposed that the help system will allow tool plugins to register a Qt-style compiled help package, local or remote HTML, or possibly a PDF document that the user can reference from within the GUI. We were able to complete all tasks for the help system, save the PDF viewer. While it was hoped to implement PDF documentation viewing during Phase I, we did not have enough time to complete more than an assessment of available libraries we could possibly use in the future.

3.5 Abstracted Visualizations

Visualizations and specialized widgets can be componentized into plugins that are loaded at runtime. This will allow the widget to be easily shared between multiple tools with a minimized memory and CPU resource footprint. We proposed several base visualizations, such as pie charts, line charts and bar charts, to be created for the PTGF and made available to tools using the framework. The abstracted data model-view mechanisms already offered by Qt are ideal for the visualization plugins in this framework, and we capitalized on these existing features.

Further we proposed to investigate the feasibility of the following visualizations using available libraries: 3D surface plots; directed graph visualization; network analysis graphs. Investigations will include work required for implementation during Phase II, which may require creating the visualization from scratch without the use of an external library.

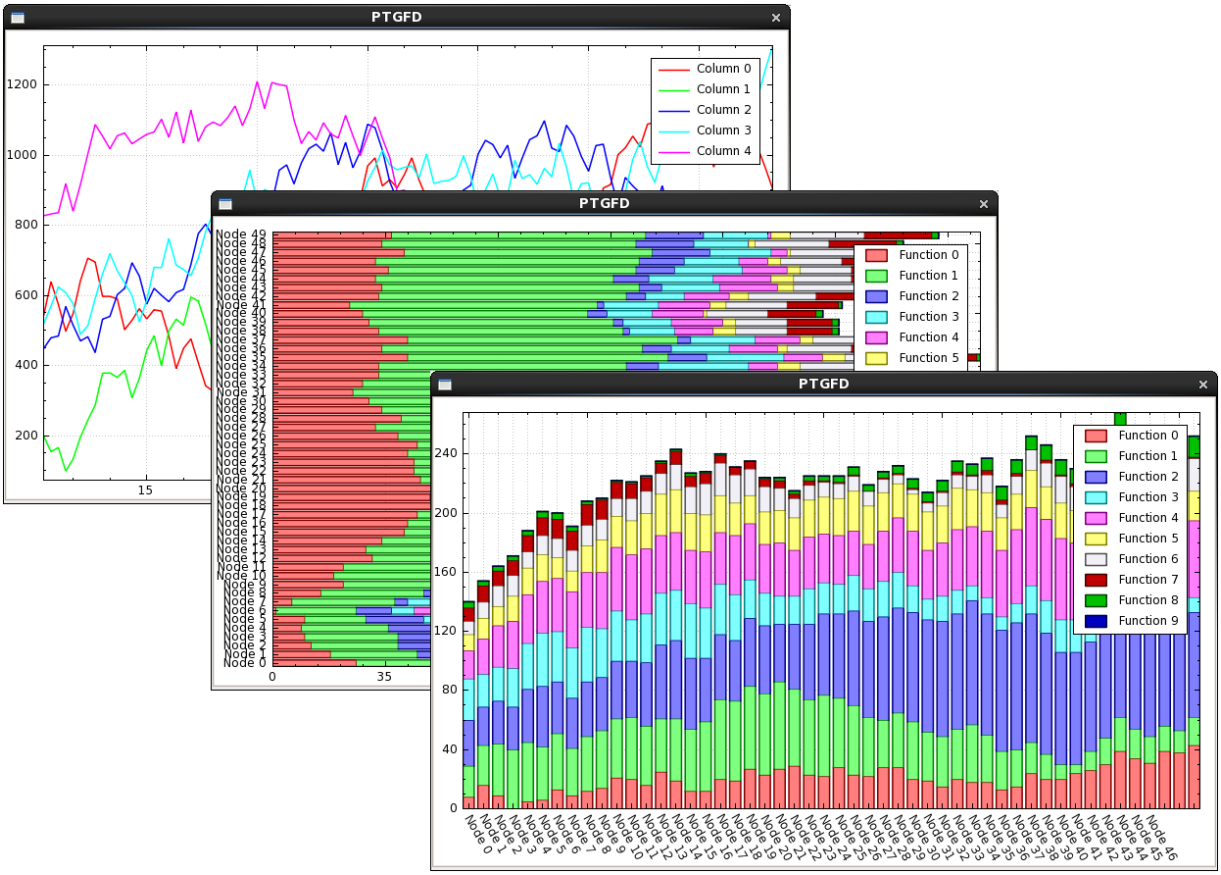

Figure 4: Visualization charts demonstrated

Using previous work on the O|SS GUI we created an abstracted ViewManager for handling views and matching them to customer's data models. For example, a tool developer can provide a populated data model to the view manager, and get a list of compatible views in return. From there the developer can select the most appropriate view, and present it to the user.

During Phase I we extended Qt's default table view to handle a wider variety of data types, like percentages. We were also able to create an abstracted source code viewer and annotation widget, which allows the user to see collected data alongside their source code. Created 2D views include column chart, bar chart and line chart views (Figure 4, and Figure 1).

Unfortunately, a suitable visualization library for simple pie charts and scatter plots were not found, and the visualization work will have to be performed from scratch at a future time.

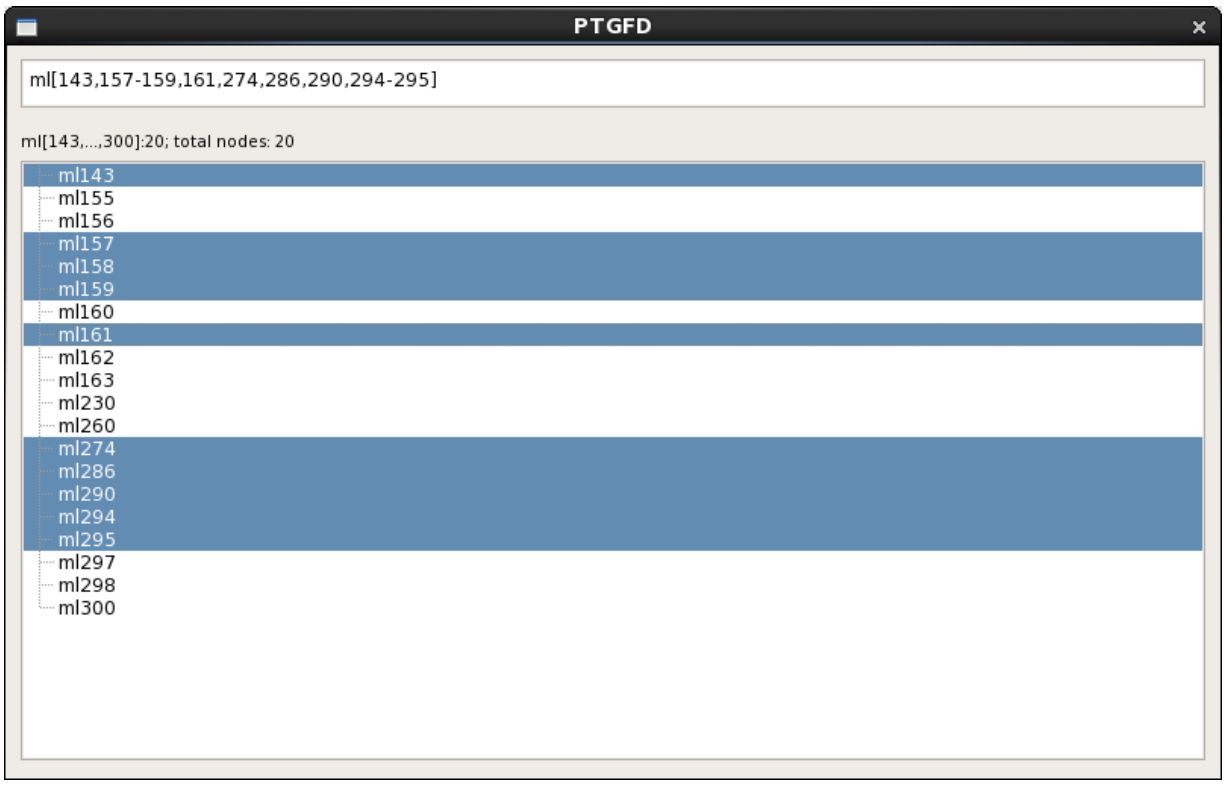

Figure 5: Node List Widget demonstrated

Although work for a node list widget was performed under funds from another project, we think it worth mentioning here. The node list widget allows the user to manually select from a list of nodes using the mouse, or using a textual representation of a node list. This node list is currently automatically populated by job allocation on Slurm machines. Given this, a user can manually select nodes 1,2,3,5, and 10 using the mouse, or could type "1-3,5,10" into a text box. This allows both novice and expert users to quickly select node sets for use with a tool. See Figure 5 for a screenshot.

3.6 Network Update Plugin

The choice of working with a runtime loading plugin architecture allows for the possibility of on-the-fly updating of individual components through a networked updating system, further easing system administration. We proposed to assess, during Phase I, the possibility of including this feature in future work. As explained previously in at the end of Section 2.2, after several conversations with DOE laboratory personnel it was found that, while this feature could technically have a place in their HPC production environments, network security policies would likely prevent a feature like this from actually being used.

4 Products Developed

In this section we identify PTGF related products developed under the Phase I award and technology transfer activities[5].

4.1 Public Software Release

A public release of the Parallel Tools GUI Framework v0.4-alpha[5].

4.2 Public Presentations and Demonstrations

Technology demonstrations, and a slide show of the Parallel Tools GUI Framework were given to attendees of the annual Supercomputing 2013 conference at the Open|SpeedShop booth [3].

4.3 Websites

A website detailing PTGF features and exhibiting screenshots was created to reach out to potential customers and users, <http://www.paralleltoolsguiframework.com/>. The source code and API documentation can be accessed publicly at the project's current public repositories, <https://github.com/PTGF>.

4.4 Collaborations

The PTGF is currently being used by several open source projects, and Argo Navis Technologies is actively working toward expanding the adoption of the framework as other tool's user interfaces. Current tools are as follows:

- Open|SpeedShop[4] [Krell Institute]
- Stackwalker Analysis Tool[6] (SWAT) [University of Wisconsin]
- CBTF DaemonTool [LANL], a suite containing the following individual parallel tools:
    - psTool
    - memTool

5 References

[1] J. McDonald. (2010, Feb. 15) *Qt 4.6.2 Released* [Online]. Available: http://blog.qt.digia.com/blog/2010/02/15/qt-462-released/

[2] A. Somers, Nokia. (2011, Dec. 15) “Model/View Programming,” *Qt 4.7 Project Documentation* [Online]. Available: http://qt-project.org/doc/qt-4.7/model-view-programming.html

[3] D. Gardner. (2013, Dec. 4) *Supercomputing 2013 Followup* [Online]. Available: http://www.paralleltoolsguiframework.com/sc13-followup/

[4] Krell Institute. (2013) *Open|SpeedShop Overview* [Online]. Available: http://www.openspeedshop.org

[5] D. Gardner. (2013, Dec. 8) *Parallel Tools GUI Framework Public Release 0.4-alpha* [Online]. Available: https://github.com/PTGF/PTGF/releases/tag/0.4.0

[6] Paradyn Tools Project, “Stackwalker Analysis Tool,” University of Wisconsin-Madison, poster presented at the Supercomputing Conference, Salt Lake City, UT, 2012. Available FTP: ftp://ftp.cs.wisc.edu/paradyn/posters/SWAT12SWAT.pdf