Active Code Generation and Visual Feedback for Scientific Workflows using Tigres

Ryan A. Rodriguez
Lawrence Berkeley National Lab
Berkeley, CA
Submitted to the TRUST REU
Email: {ryanrodriguez}@lbl.gov

***Abstract*—The focus of this research will be on the development of an experimental user interface that aims to accelerate the prototyping of workflows by providing real-time visual feedback and code generation functionality. Scientists prefer to work in code, but today's workflow tools have a graphical focus. It was hypothesized that an effective interface for composing workflows would be able to leverage the flexibility of coding while still providing real-time visual interactivity. Secondarily the interface could accelerate analyses by generating the code necessary to execute workflows. The Tigres API for next-generation workflow production calls for precisely this type of interface.**

The Tigres project is a next generation scientific workflow API designed with ease of composition, scalability and light-weight execution in mind. The experimental interface known as the Active Console has been designed to meet these challenges. Inspired by Google's real-time search bar and IDEs that allow for in-line execution, its intent is to allow for minimally intrusive execution and maximum flexibility. Using Python GUI capabilities alongside specialized visualization tools, it is possible to implement the three pillars of the interface, which are the real-time visualization tool, the automatic code generation feature, and the console itself. The result is an interface that feels like an IDE, but provides the option of graphical interaction and compositional streamlining.

I. Introduction

Researchers in computationally intensive sciences are faced with analyzing rapidly expanding data sets on increasingly parallelized machines. These datasets have grown so large that it is no longer practical for partner institutions to download and analyze this information locally. This poses challenges for collaborators who wish to run data-intensive workflows at remote supercomputing centers. Although software solutions exist to address some these concerns, confusing interfaces and specialized syntaxes have made the cost of integration into existing projects prohibitive. The Tigres API aims to rectify these usability issues by providing a programming library that fits easily into existing work. Because Tigres is an API first and foremost, it currently lacks the visual component of other tools. Therefore, the development of an alternative interface that can manage interactivity while leveraging the flexibility of tigress is of great interest.

A workflow tool with the flexibility of Tigres in addiiton to an interface designed to accelerate workflow composition could dramatically simplify data intensive research. With Tigres' unique position as an API amongst primarily graphical tools, it holds great promise for briding the gap between code and visual feedback. Currently, scientists building workflows have one of two options: they can choose to write their own scripts in a programming language of their choice, or they can employ a specialized workflow tool like Taverna [1], Pegasus [2], or Kepler [3]. By taking the programming option, a researcher is allowed the maximum flexibility and compatibility, but faces significant overhead in dealing with parallel computing challenges. Alternatively, researchers choosing the GUI option for workflow design forego flexibility for ease of workflow composition. Despite high demand for capable workflow tools, existing applications have failed to gain widespread acceptance due to interfaces which do not make a clear connection between coding and visualization.

Before work on a new interface began, the Tigres graphing module relied on the execution of a workflow. This 'execution only' model is ineffective for composition time visualization and prototyping of workflows. Additionally, the ability to generate and update code dynamically based on graphical or textual input is an area of ongoing research. The development of an interface that focuses on the tight integration between coding and interactivity poses challenges not only in computer science, but in API design.

At its core, Tigres endeavors to be as minimally intrusive as possible. A researcher using Tigres in the IDE of their choice should find that this experience is indistinguishable from using their IDE on any other type of project. This is desirable because users have expressed their discontent with confusing features and opaque interfaces. The focus of this paper is the design and implementation of an interface developed to engage users and accelerate workflow development. This work aims to be a proof of concept for a future integrated environment where users can move seamlessly between graphical and source code domains. This envisioned 'Active Console' would be the link between an interactive graphical interface for creating workflows and a code generation feature.

The rest of the paper is organized as follows. A brief background on the Tigres project is followed by a discussion of related work. Section III covers specifics of the interface and section IV contains the conclusion of the research so far.

II. Background

A. Tigres

A popular approach to dealing with data-intensive tasks has been to follow the MapReduce pattern for parallelization, merging, and splitting tasks - sometimes referred to as 'Scatter-Gather.' At present, writing scientific workflows using a MapReduce model with a Hadoop implementation

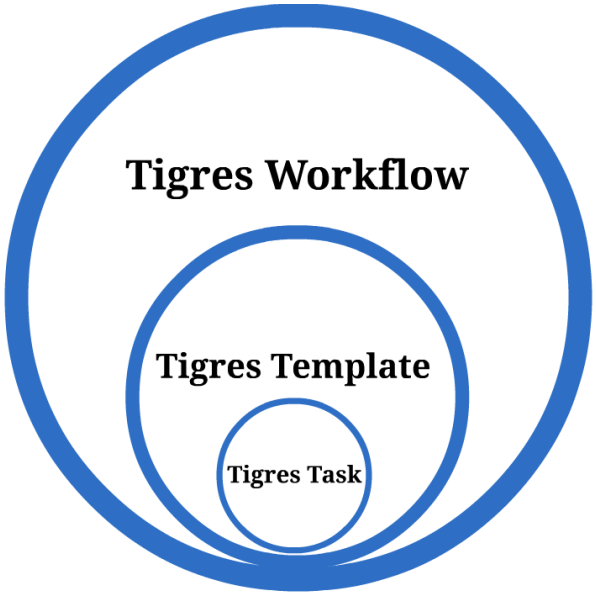

Fig. 1: The Tigres hierarchy

can require a significant amount of programming expertise and custom code. Some difficulty arises in scientific workflow implementations stemming from the difference in the nature of typical MapReduce tasks in comparison to scientific functions. Additionally, writing scripts from scratch comes with a considerable amount of overhead due to the complexity of provenance tracking, parallel fault tolerance and monitoring features. Currently there is a significant separation between workflow implementations in programming languages, and those built using specialized GUI workflow tools. In light of these challenges, a tool that can simplify the implementation of common computing patterns while offering a high degree of interactivity is highly desirable.

The goal of the Tigres project is to develop a next generation scientific workflow tool that addresses these concerns. Following the examples of the MapReduce model, in particular the Apache Hadoop implementation, Tigres is designed to be a highly portable and lightweight API to radically simplify the deployment of scientific workflows to diverse resource environments. Workflows in Tigres can be composed with the flexibility and ease of standard technologies like C and Python while eliminating the overhead of writing custom workflow scripts. To use Tigres, a user simply needs to import the Tigres library into their project and compose their workflow using Tigres Templates1. These templates represent the most commonly used computational patterns: Sequence, Parallel, Split and Merge. These templates are used to organize Tigres Tasks, the atomic unit of execution. Each Task object holds a function or executable, input values and input types. In constructing a Workflow using these Tigres types, a user has done everything that they need to invoke the advanced monitoring features of Tigres. Provenance tracking, fault-tolerance, execution-state and visualization tools become artifacts of Tigres' use, not overhead. This model allows analyses created on a desktop to be scaled and executed seamlessly on a diverse set of resources.

B. Related Work

The demand for applications which address issues facing e-Science has grown dramatically in recent years. Several workflow tools have been developed to address this demand. Discovery Net [4] was one of the first tools available to coordinate the execution of remote services using the web. It provides a means to coordinate workflows between data owners and analysis servers that may be far-flung geographically. The system also allowed for visual coding through a drag and drop interface. Although the Discovery Net project was quick to recognize the need for workflow coordination services, their XML based language known as 'Discovery Markup Language' in addition to its domain specific nature and high cost of integration into existing projects did not allow for widespread adoption. Taverna Workbench [1] is an open source tool that has emerged with a graphical focus; using a limited drag and drop functionality, users can construct a workflow with representations for inputs, outputs and various services. The application facilitates the building, execution, and collaboration of workflows through web services. The interface provides a services tab to associate graphical blocks with analytical functions, and a standard project hierarchy navigator. Coined as an 'enactment engine,' Taverna's integration with popular programming languages is difficult. Additionally, users quickly find themselves lost in a maze of drop down windows.

Kepler [3] is another open source GUI based workflow tool being developed as a joint project between multiple University of California campuses. It offers support for various grid services and has the ability to execute workflows from a command line or graphical interface. Like Taverna, however, Kepler's main drawback is its loose association with C or Python. Instead, it uses a host of previously written components for statistical analysis.

Triana's [5] take on workflow development was to formulate a 'pluggable architecture' that can integrate into existing systems. It comes with a variety of built in tools for signal analysis as well as image manipulation.Triana only supports basic error detection. Again, the drawback of this tool is the user is confined to using a set of pre-made functions. If a custom function is desired, it must be specified in a wizard provided by the application.

The concept for Pegasus [2] is interesting in its divergence from the others. Instead of the typical graphical focus, Pegasus' focus is on being able to execute on a wide variety of distributed environments including clusters, grids and clouds. Pegasus has the ability to map elements of a given workflow to available resources on the target environment. Pegasus offers a GUI, but goes so far as to include a disclaimer on their site which reads, "It is a demonstration tool and can be used to see how simple DAXes look like. However, to generate your workflows, we recommend to use the various APIs we provide." The broad class of computational patterns (DAXes) that Pegasus supports makes the API powerful, but non-trivial to learn or use.

Within these projects, there is little support for automatic scaling to fit growing datasets. Further, the problems of fault tolerance and dynamic workflow execution remain. Focus on exotic computation patterns or specialized syntax often makes for confusing APIs and limited usability.

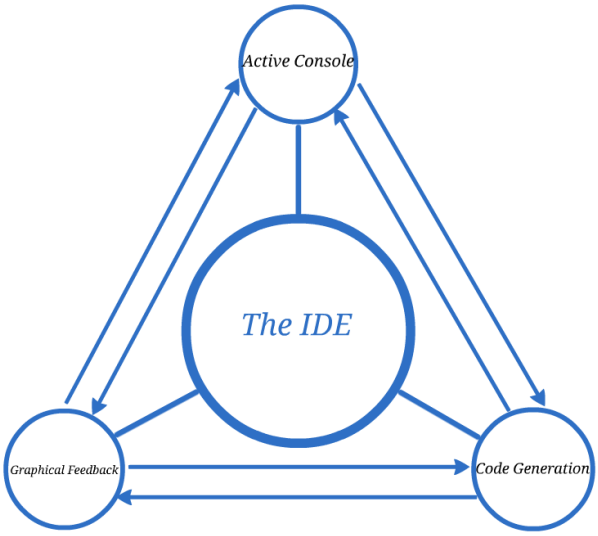

Fig. 2: Structure of the Active Console

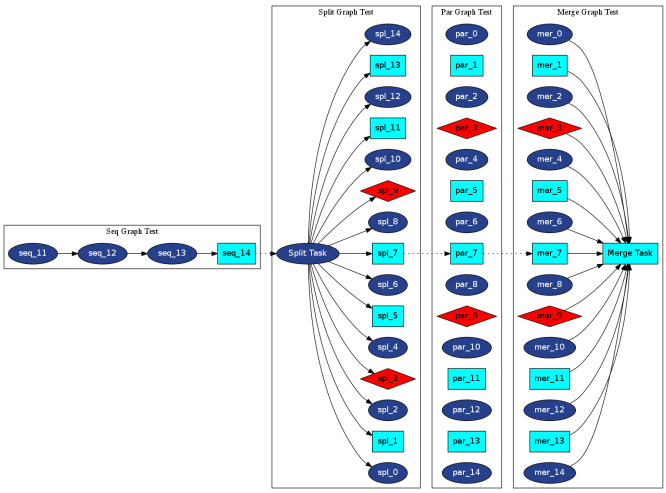

Fig. 3: A Tigres workflow shown in Dot

III. Active Console

The Active Console is the experimental visual and textual interface for composing Tigres workflows. The interface consists of three main components: the console, code generator, and GUI for manipulating workflows2. The user has the option of running the Active Console and its graphing feature from the IDE. From here, the user will have the option of specifying a workflow textually or graphically, while receiving the code necessary to run the workflow they've just specified. Changes to the workflow specification in either the graphic or text domains result in an update of the other in real-time.

In order to visualize the workflows being specified, the costs and benefits of the Graphviz and the D3 javascript library were studied. Graphviz was employed for the initial production of workflow execution graphics. Graphviz is an open source tool built on top of several popular graphing engines including 'dot', 'neato', and 'sfdp'. 'Dot' is a tool built by AT&T for representing complex networks. It provides edge drawing and placement algorithms that allow dot files to be specified quite simply. Additionally, 'dot' offers a wide variety of sub graphing and rank specifications for exhaustively specified graphs. The tool can output into a number of common image formats including SVG, making manipulation of 'dot' files possible in web pages. Currently, 'dot' graphs take the form of static output images. There is little *direct* support for graphical editing.

This functionality is built into the monitoring capabilities of Tigres. In other words, generation of workflow execution graphs is another artifact of using Tigres types to construct a workflow. Initially, this functionality has been limited to working on executions from memory. Future iterations of the Tigres graph module can work from log files which track Tigres execution on computing systems in real-time. This is a critical extension because graphing from log files will allows real-time tracking when executed on a HPC system. Each task holds information about its current execution state, allowing for changing visuals depending on the current status of a task.

While direct support for interactive graphics in Dot is not possible, web browsers offer the advantage of advanced visual capacity without additional software tools. The first choice for cross platform compatibility and visual flare was the D3 [6] library in javascript. JS Plumb [7], GraphDracula [8], and Arbor JS' HalfViz [9] were considered for building interactivity, but offered no clear advantage to D3. D3's focus is on data-driven documents; by binding visual elements in a web page to data elements it is possible to create dynamic graphs and animations. Additional motivation for implementing visualization and manipulation functionality in a browser was to conform to Tigres' goal being easily adoptable. It's expected that many users would prefer to stick to their editor or IDE of choice, so having this functionality in a web browser instead of a standalone application would be less intrusive to the user. Additionally, due to the dynamic nature of the tool it would be well suited to operate on changing data sets.

Several visualization schemes were used as test cases for D3, including the a directed graph and hive plot4. Although this model is visually interesting and fun to interact with, it suffers from a 'hairball' effect for graphs of non-trivial size. To address this concern, a 'hive' visualization paradigm was explored. Hives use axes to organize nodes and draw edges based on user defined metrics. A secondary motivation for this style of visualization was the difficulties posed when representing complex dependency networks in a workflow. The 'hairball' effect when representing graphs in Dot and D3 were compounded when edges showing their dependencies were added. In a hive format, these 'hairballs' took an orderly and visually pleasing form. Hive graphs solve the problem of data dependency visualization, but are not easily understood in the context of execution graphs. Although the D3 library offers a robust solution for web-based graphics. In many cases, however, it can tend toward the experimental or quixotic.

The idea for a workflow tool that could emphasize the relationship between coded workflows and their graphical output was inspired by Mathematica, which allows users to write a brief snippet of code and execute it within the editor. The output of such an execution is displayed in-line with the code and can take the form of a graphical or textual output.

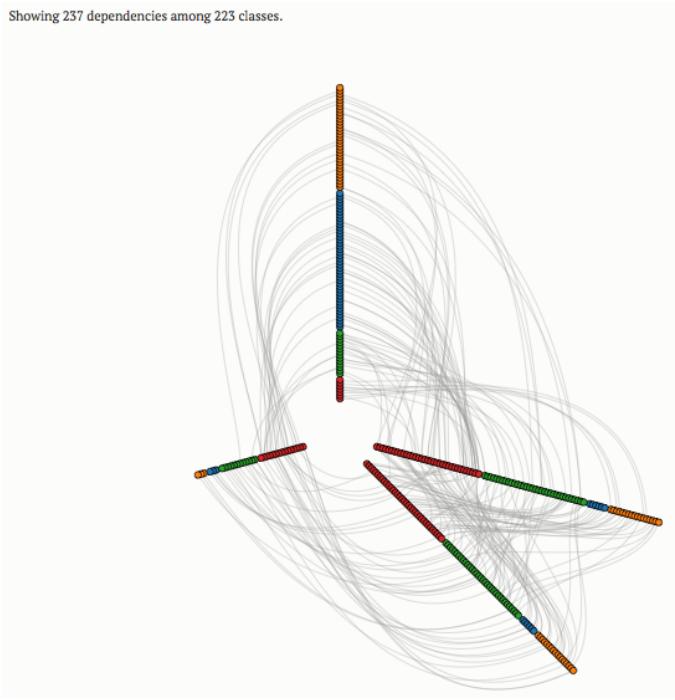

Fig. 4: Hive Plot showing data dependencies in Tigres

In order to improve the feel and response of the application, cues were also taken from the Google real-time search bar, which allows users to see responses to their queries as they type. This functionality is desirable from a user stand point because it improves the response of the interface compared to a command line where a user specifies one command at a time, or a static notepad where a user has to click a button for output.

The console is, as the name implies, a no frills text editor for passing the entire Active Console system workflow commands. The Active Console is the addition of graphical and automatic coding features to this console, shown in Fig.5. The console runs multiple threads on each update to search for any of the Four Tigres Teplate keywords: Sequence, Split, Template or Merge. Once these are located, the editor parses the task list and recurrence list arguments which specify the task, and how many. The editor also looks for a 'generate' or 'exit' command. With only six commands, four of them having function and recurrence arguments, a user can completely specify complicated workflows textually and get visual feedback immediately.

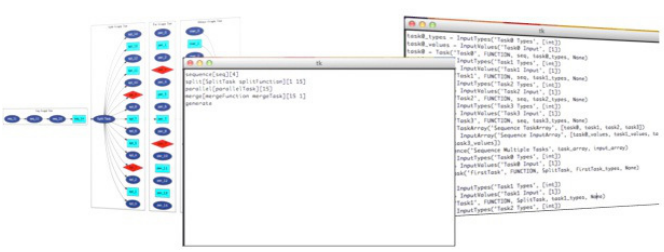

Fig. 5: The Active Console Interface

Several open source IDEs built entirely in Python currently exist which could be extended to include the 'Active Notepad' functionality, along with other features of a modern Python editor. For the purpose of UI research, the Active Notepad is a sufficient test bed. Python's built-in GUI library, TkInter, can produce the desired interface components including the active notepad, and code generation output area. However, getting Dot output to display in a TkInter window is not a straightforward task. To achieve a mock-up for future interactive capability, a library called XDot [10] is used of displaying Dot files in a separate window. Although this library does not currently offer the ability to modify graph by dragging and dropping nodes, It holds considerable promise for future development of more interactive features.

In parallel with the console and graphics capabilities of the Active Console, the code generation feature will use the input from either the console, or the interactive GUI representing the workflow, to produce all of the code necessary to run the specified workflow using the Tigres API. This has the potential to provide a significant productivity advantage over traditional coding methods.

In the process of visualization research, many hurdles involved with implementation specifics were cleared. However, the issue of being able to specify workflows graphically has proven to be difficult outside of a monolithic application. Presently it is not possible to specify workflows graphically, although strides have been made using modern browser capabilities. Currently a user can drag and drop nodes and edges, delete nodes, and include or exclude a task from a template, but it is not yet possible for tasks to be created. The problem of getting these updates to translate to updated workflows in Python also remains. Once full editing functionality has been achieved, it will be possible to link the console and GUI so that editing one updates the other.

IV. Conclusion

The purpose of this research was to investigate the possibility of an interface that provides visual feedback and workflow composition capability while remaining minimally intrusive. For the initial test cases, ATT's Dot language was employed to output graphs generated by the execution of Tigres Workflows. To extend this capability, various possibilities for making graphs interactive and editable were explored. D3 has proven to be a very capable library for the construction of dynamic and interactive graphs. However, the challenge of editing and interpreting these edits in D3 remains. Additionally, users may find the flashy nature of the library distracting. Additional research has uncovered a method of applying a jQuery plugin to SVG output of Dot files which may hold the greatest promise moving forward.

Through careful and honest analysis of the existing graphical workflow tools, the need for a more direct connection between graphic and programming domains was discovered. By employing built in Python libraries to build GUI elements which can be executed alongside existing workflows, the focus on the IDE remains paramount while the interface is available on demand. This paradigm leverages the flexibility and feature set available when coding in a modern IDE while preserving the visual capability of other workflow tools.

Although graphical interactivity has remained somewhat elusive, the underlying principle of the Active Console has proved to be quite promising. The reduced command set and code generation features can dramatically reduce time from concept to workflow execution. Unlike the current generation of tools that draw a project into a box, the Tigres API with the Active console will give the user ultimate freedom through complete control.

ACKNOWLEDGMENT

This work was supported in part by TRUST, Team for Research in Ubiquitous Secure Technology, which receives support from the National Science Foundation (NSF award number CCF-0424422).

Additionally, this work was funded by the Office of Science of the U.S. Department of Energy under Contract Number DE-AC02-05CH11231.

REFERENCES

[1] T. Oinn *et al.*, "Taverna: lessons in creating a workflow environment for the life sciences," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1067–1100, 2006.

[2] E. Deelman, G. Singh, M. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. Berriman, J. Good, *et al.*, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, vol. 13, no. 3, pp. 219–237, 2005.

[3] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludscher, and S. Mock, "Kepler: An extensible system for design and execution of scientific workflows," in *Proceedings of the 16th International Conference on Scientific and Statistical Database Management*, pp. 423–424, 2004.

[4] S. AlSairafi, F.-S. Emmanouil, M. Ghanem, N. Giannadakis, Y. Guo, D. Kalaitzopoulos, M. Osmond, A. Rowe, J. Syed, and P. Wendel, "The design of discovery net: Towards open grid services for knowledge discovery," *International Journal of High Performance Computing Applications*, vol. 17, no. 3, pp. 297–315, 2003.

[5] I. Taylor, M. Shields, I. Wang, and A. Harrison, "Visual grid workflow in Triana," *Journal of Grid Computing*, vol. 3, no. 34, pp. 153–169, 2005.

[6] "D3 data driven documents," 2013.

[7] "jsPlumb api," 2013.

[8] "Graph dracula website," 2012.

[9] "Arbor js halfviz website," 2012.

[10] "xdot website," 2013.

[11] "dot jquery plugin website," 2013.

[12] P. Nguyen and M. Halem, "A mapreduce workflow system for architecting scientific data intensive applications," in *Proceeding of the 2nd international workshop on Software engineering for cloud computing*, SECLOUD '11, (New York, NY, USA), pp. 57–63, ACM, 2011.

[13] E. Dede, M. Govindaraju, D. Gunter, and L. Ramakrishnan, "Riding the elephant: managing ensembles with hadoop," in *Proceedings of the 2011 ACM international workshop on Many task computing on grids and supercomputers*, pp. 49–58, 2011.

[14] "Tigres website," 2013.

[15] "Graphviz website," 2006.

[16] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, "The landscape of parallel computing research: A view from berkeley," tech. rep., EECS Department, University of California, Berkeley, Dec 2006.