Development of a Distributed MATLAB Environment with Real Time Data Visualization

Austin McEver
College of Electrical Engineering and Computer Science
University of Tennessee, Knoxville
Knoxville, Tennessee 37916
Email: rmcever@vols.utk.edu

Joseph Diamond
College of Electrical Engineering and Computer Science
University of Tennessee, Knoxville
Knoxville, Tennessee 37916
Email: jdiamon3@vols.utk.edu

Mohammad Ahmadzadeh
College of Electrical Engineering and Computer Science
University of Tennessee, Knoxville
Knoxville, Tennessee 37916
Email: mahmadza@vols.utk.edu

***Abstract*—The need to communicate among multiple programming languages and simulators inevitably arises in any large system simulation. CURENT's Large-Scale System Testbed (LSTB) is no exception. Its numerous MATLAB and Python simulators and modules require the ability to send variables and otherwise communicate with one another. This research project develops a scheme to meet the LSTB's need for such communication. By building a server on top of the existing Python-MATLAB-Bridge project and creating APIs for both MATLAB and Python, we have successfully developed a scheme to allow easy, efficient communication between the two. Additionally, we are developing a graphical user interface (GUI) in Python that communicates with the same server to visualize the results of the simulation.**

I. Introduction

The work discussed in this paper concerns the development of a distributed MATLAB environment that allows communication among many instances of MATLAB and many instances of Python. While this software was tailored to the needs of CURENT's Large-Scale System Testbed, it could be applied anywhere different instances of MATLAB and Python need to share variables and otherwise communicate with one another. A server sits at the center of the environment, handling all client requests and saving messages until a client requests to receive its respective messages. In addition to the distributed environment, this paper discusses the beginnings of the development of a visualization module that also connect to the server to graph and display the results of the Large-Scale System Testbed simulations and eventually to send back controls to the system. This is all necessary because MATLAB is inherently single-threaded and cannot quickly or efficiently share its workspace variables with different instances.

A. Context of the Large-Scale System Testbed

The purpose of the Large-Scale System Testbed (LSTB) is to simulate the power grid of the future and anticipate its needs and behaviors. To accomplish this goal, the LSTB employs a variety of simulations across multiple computer systems to mimic the busses of power systems. The majority of the modules within the grid use MATLAB in some form, and these modules need to communicate with each other to fully control and implement the grid simulation. MATLAB workspaces, as they come in the base installation, do not facilitate the sharing of data between multiple MATLAB workspaces. Therefore, we needed to make a platform that lets MATLAB clients share their workspaces and allows easy addition and subtraction of modules from the simulation system.

Additionally, these simulations generate large amounts of data at rapid speed, which makes interpreting the raw data difficult for humans. Graphing this data in real-time helps present the information in the most convenient form to ultimately realize the goal of the LSTB project.

B. Background Research

While there are a few options available to create a distributed MATLAB environment functionally similar to the one created through this project, none of them quite fulfilled the needs of the LSTB. The Python-MATLAB-Bridge (Pymatbridge) project on Github provides a way to communicate between Python and MATLAB, but its purpose was not to connect MATLAB clients via Python, but to essentially run MATLAB code in Python. Since the LSTB modules were already written and ready in MATLAB, the Pymatbridge did not fulfill the needs of the LSTB on its own (though we did build our distributed system on top of Pymatbridge).

The new MATLAB engine, which has functionality similar to the Pymatbridge, provided another potential option. It allows programmers to share data and run MATLAB computations in C, C++, Fortran, and Python. Despite its formidable functionality, MATLAB engine is only available for MATLAB 2014b and newer. Since some of the modules of the LSTB were written in older versions of MATLAB, this option could not be implemented, though it did find its place in the test scripts written to accompany the distributed system.

The MATLAB Distributed Computing Server provided some of the functionality necessary to satisfy the LSTB, but its cost and superfluous features persuaded the group to seek another option. It also lacked a method for communicating with other languages, which would have inhibited the development of a visualization module outside of MATLAB.

Quanser Real Time Control (QUARC) communications protocol provides a method for sharing MATLAB workspaces, but it is primarily implemented for singular machines. This disqualified QUARC for usage along with its inability to

efficiently and effectively share variables with other languages, but it did provide some ideas with how the distributed environment might work.

C. Project Description

The LSTB runs a persistent Virtual Grid Simulator (VGS) that needs to communicate with analysis tools, such as Power System Analysis Toolbox (PSAT) and ePHASORsim. Figure 1 provides a visual overview of the LSTB behavior. The VGS sends its data to the Streaming Server which then forwards the data to the other connected modules. Then modules receive the data and apply state estimation, contingency screening, and voltage controls to the grid data and examine the responses. State estimation modules attempt to minimize the error of power grid measurements and replicate the exact state of the power grid. Contingency screening modules, meanwhile, apply hypothetical contingencies to the data and attempt to resolve the contingency with the grid's current state. Finally, the Voltage Control module detects voltage irregularities and attempts to maintain the voltage via shedding load or other methods. Since all of these modules need to share data with each other, this project attempts to build a platform to facilitate communication among these modules.

An additional part of this project involved creating a visual representation of the data from the LSTB in real-time, which requires three primary components: modules must send the Time Domain Simulation data off to a server, a Streaming Server must receive, transfer, and process this data, and finally a Visualization module must plot the data into a human-readable graph. Since these modules incorporate various softwares, the distributed environment must be modular and flexible. To fulfill all the parameters, we assisted in the development of the Distributed MATLAB Environment (DiME) software.

The DiME software constitutes the Streaming Server shown in figure 1 and the graphical user interface is synonymous with the Visualization module shown. Since our work thus far has primarily regarded the Streaming Server and Visualization modules, the rest of this paper discusses these two subjects in detail.

D. Software Layout

DiME relies on a few key modules that allow it to provide an environment that facilitates the communication of modules across multiple instances of MATLAB and Python. The Python-MATLAB-Bridge (Pymatbridge) interface provides a basis for communication between MATLAB and Python. It employs ZMQ, a software package that provides programmers with an easy way to send and receive messages in different languages, to facilitate communications between Python and MATLAB. Pymatbridge also relies on NumPy, a package for scientific computing in Python, and JavaScript Object Notation (JSON), a data interchange format, to encode, decode, and send variables among modules. These dependencies along with a server script, Python and MATLAB APIs, and a set of test scripts constitute the DiME software suite.

The server acts as a liaison for each instance of MATLAB and Python. It waits for instances to connect, keeps a list of connected clients, waits for clients to sync their messages, and contains a cache made up of a collection of queues, one for each device. When a module sends a variable, it either sends it to a specific module by name or broadcasts the variable to all other modules. The former adds the variable to the appropriate module's queue on the server, while the latter adds the variable to every queue on the server. When a module syncs, it gets the next message from the queue. See figure 2 for a visual description of the server's request response patterns.

II. Development of the Distributed MATLAB Environment (DiME)

The LSTB consists of several modules in MATLAB and other simulation environments. To facilitate an efficient simulation, these modules need the ability to share information and communicate with one another. DiME provides a solution to this problem.

A. Implementation

Clients connect to a server via sockets implemented through the ZeroMQ API. Once connected, the clients can send or broadcast variables to the other clients through the server. The server uses a router-dealer socket configuration to efficiently transfer data from clients to worker threads on the server and back to clients. This configuration prevents race conditions from occurring as the server rapidly sends and receives data to and from its connected clients.

When a client sends variables, the router socket on the server receives the message and forwards it to its dealer counterpart. Then, the dealer chooses a worker thread from a pool to receive the variable and perform the requested operation on the data. In a broadcast command, for example, the worker thread would take the variables it received from the dealer and place them in each of the connected clients queues.

When a client requests data from the server, a similar process happens in reverse. If a client sends a sync request, the router tells the dealer that it needs the variables in the client's queue. The dealer then selects a thread to pull the data from the client's queue, hands the data to the router, which then forwards the data to the appropriate client. This process is described visually in figure 2.

Initially, our design of the client/server configuration did not involve the router-dealer socket construct nor the pool of worker threads. At first, modules communicated with the server through simple request sockets on the client side and a single router socket on the server side. This, however, caused the router to directly interact with threads without the use of memory barriers, and since sockets are not thread-safe, race conditions occurred. These race conditions would break the lock-step request-reply pattern between clients and servers, causing the software on the clients and server to stop all operations as each side waited infinitely for a reply that would never come. Therefore, we changed the server side socket

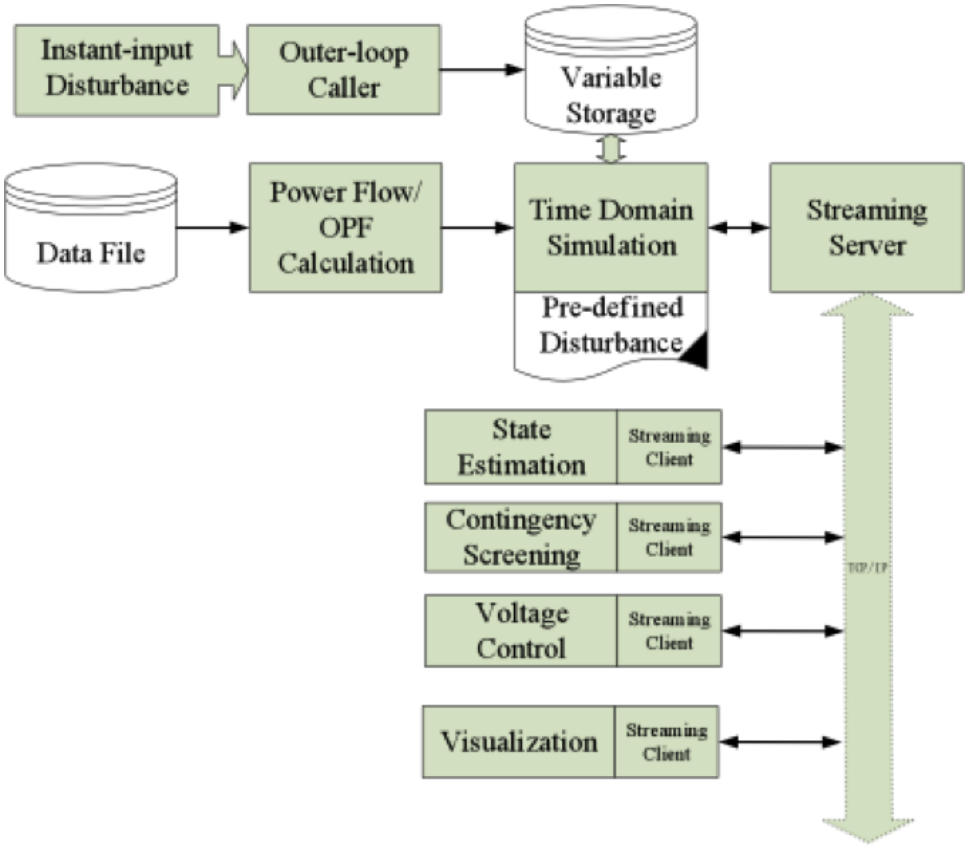

Fig. 1. Overview of the LSTB

configuration to include a dealer socket to interact with the threads as suggested by the ZMQ documentation.

B. Usage

This implementation of a distributed environment not only strives for efficiency, but it also aims to provide a simple API to allow programmers to easily connect their instances of Python and MATLAB. To connect a module, one must simply import the DiME module, instantiate a dime object with its name and potentially the server address to which it will be connecting, and call start. In MATLAB, it is also necessary to start json manually. The following code snippets provide an example of this process in both MATLAB and Python.

MATLAB

```
>> addpath(genpath('/Path/to/DiME-Module'))
>> json_startup
>> dime-object = dime(?Instance-Name?,
    [?server-address?])
>> dime-object.start()
```

Python

```
>>> import dime
>>> dime-object = dime.Dime('Instance-Name',
    ['server-address'])
>>> dime-object.start()
```

The following table describes all of DiME's provided functions.

Function	Description
Dime(name, [server-address])	class constructor, initializes the instance name and server address
start()	connects to DiME server
sendVar(recepient-name, var1, [var2], [...])	sends specified variables to 'recepient-name'
broadcast(var1, [var2], [...])	sends specified variables to all connected clients
sync([max-messages])	gets 'max-messages' from queue, gets 3 if no arguments given
getDevices()	returns a list of connected clients
exit()	disconnects and ends session with server

C. Test Scripts with MATLAB Engine

While MATLAB engine could not be used to directly implement the variable sharing among MATLAB instances due to version issues, it did find its place in DiME's test scripts. MATLAB engine made it easy to write test scripts in Python that opened multiple instances of MATLAB and

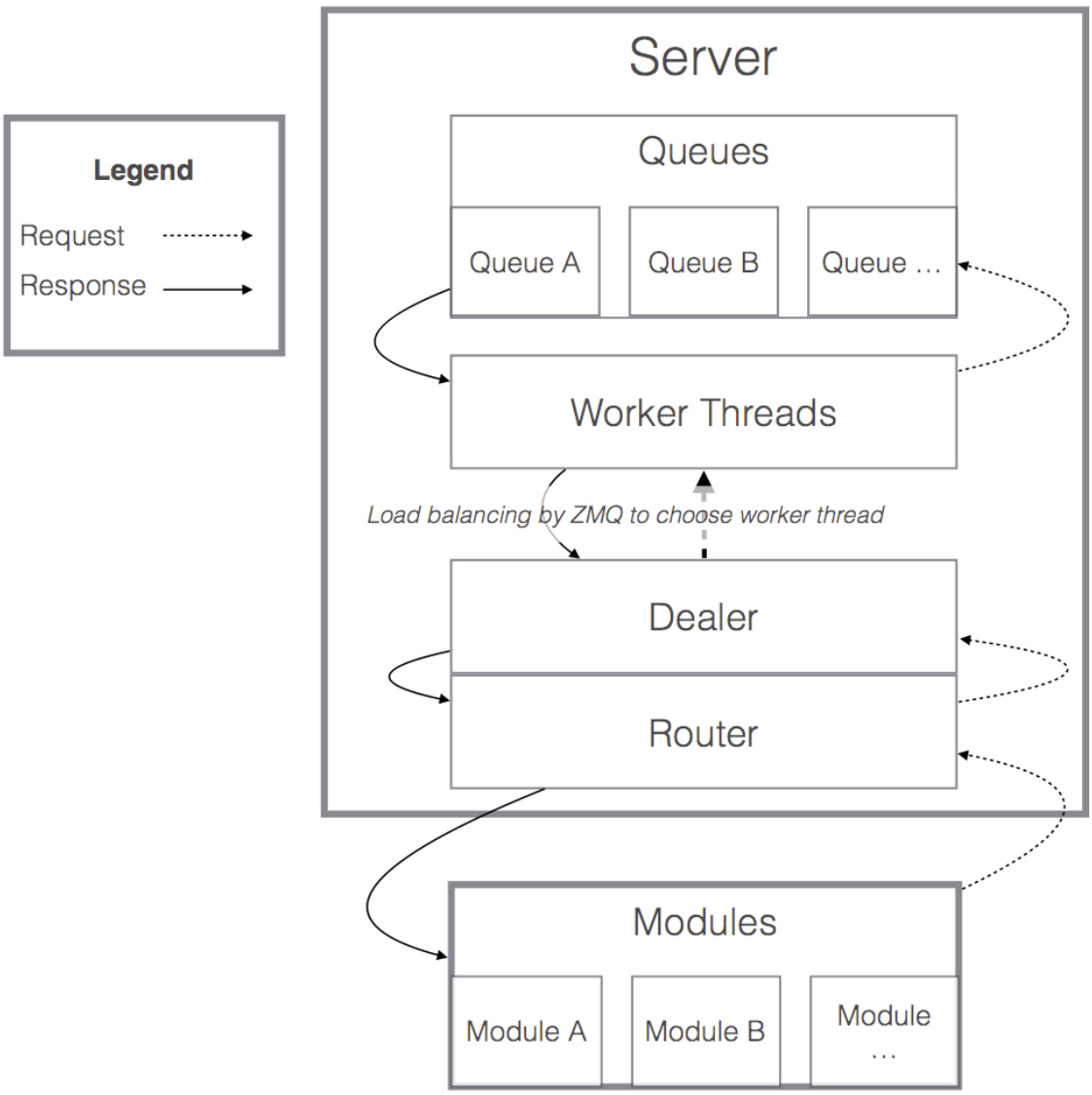

Fig. 2. Request Response Pattern in the DiME Server

had them communicate via DiME. The engine's ability to run MATLAB scripts allowed us to write whatever was convenient in MATLAB there and what was convenient in Python in the Python scripts where the entire test was implemented in one easy to run script.

III. Development of the DiME Graphical User Interface (GUI) for Real Time Data Visualization

The DiME GUI helps to make sense of the data coming from power grid simulation software through data visualization in real-time. To achieve this real-time plotting, the GUI is lightweight and flexible - it accepts variables from either MATLAB or Python-based modules. Additionally, it allows users to select which device's data should be plotted or graphed as a snapshot of time in order to examine both the big picture of the grid simulation and the finer details.

A. Dependencies

The visual elements of the GUI rely on two major Python modules: wxPython and wxmplot. wxPython is a well-rounded, general-purpose Python module for building GUIs. wxPython provided a basis for arrange the visual aspects of the

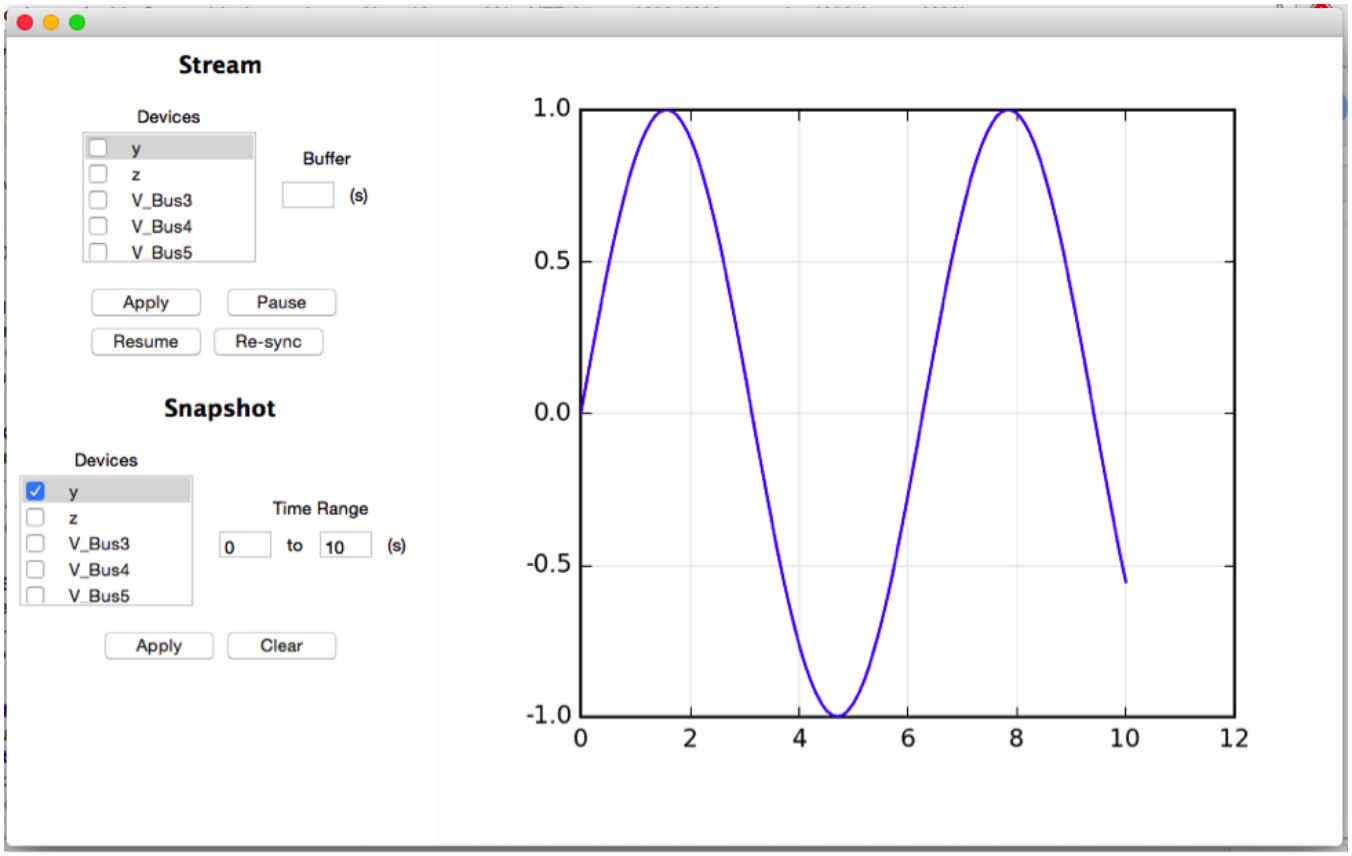

Fig. 3. Screenshot of GUI as it appears at the time of drafting

GUI (i.e. the buttons, checkbox panels, etc.) and linking them to event-driven functions. The plotting window of the GUI uses wxmplot and its underlying matplotlib implementations to create the graphs of the power grid simulation data. The result is a specialized graphical front-end for the Large-Scale System Testbed control modules that is compatible with other modules and can be adapted to interact with even more software.

B. Operation

The plotting GUI operates in either ”stream” or ”snapshot” mode. Stream mode accepts data from the DiME server and plots it in real-time while the snapshot mode allows the user to examine the data for a user-specified interval of time after it has been streamed. In both of these modes, the user can choose which devices' data should be displayed from a list of devices currently connected to the server.

Exclusive to the streaming mode, users can instruct the GUI to pause the data visualization and resume from the same timestamp at which it was paused. Alternatively, the GUI can resync the plotting to the most current timestamp. If the graphing seems choppy in real-time, users can specify a time buffer to improve graphing fluidity.

C. Current Status

At the time of this paper's drafting, the DiME server and APIs have been completed, but the visualization module is still under construction. It must be adjusted to accommodate the format of the data coming from the simulation, and the graph must be edited to include axes labels, a title, a legend, and more. Menu functionality that allows the user to export the graph and data will be also be added. See figure 3 for a screenshot of the GUI as it currently appears.

IV. Future Work

Future work for this project will focus on completing the visualization to operate as required by the LSTB. Once the module operates to its required specifications, it will be further modified so that its functionality extends beyond only receiving data from the LSTB and visualizing the results. The module will include control functionality so that commands and data can be sent to the LSTB to make adjustments as necessary to implement the best possible simulation of large scale power systems.

Later, the DiME software suite may be upgraded to run faster and more efficiently and to support more types of modules. As of right now, DiME contains APIs for MATLAB and Python, but it is expandable such that it could include APIs for other languages that may help speed up or otherwise improve the functionality of the LSTB. Using various timing tools, it also might be possible to improve the overall performance of the server to send and receive messages even faster to improve

the fluidity of the visualization module as well as providing a more responsive and realistic simulation.

V. Conclusion

The DiME server currently operates and effectively connects instances of MATLAB and Python, allowing them to communicate with simple APIs. This functionality is being implemented in the LSTB, and the module is flexible enough to apply anywhere such functionality might be needed. The visualization module is scheduled to be completely operational by the end of the day on August 14, 2015, and will continually be upgraded as required by the LSTB project. With little adaptation, the GUI could also be applied to graphing tasks retrieving data from multiple instances of MATLAB and Python via the DiME software.

Acknowledgment

This work was supported primarily by the Engineering Research Center Program of the National Science Foundation and the Department of Energy under NSF Award Number EEC-1041877 and the CURENT Industry Partnership Program.

We would also like to thank our mentors Hantao Cui and Mohammad Ahmadzadeh for their hard work, ideas, and support. Additionally, we would like to thank our faculty advisors Dr. Kevin Tomsovic, Dr. Fran Li, Dr. Yilu Liu, and Dr. Jian Huang for their support.

References

[1] Dunn, Robin and Pasanen, Harri (2014) WxPython (Version 3.0.2.0) [Computer Program]. Available at http://www.wxpython.org/ (Accessed 22 July 2015).

[2] The MathWorks Inc. (2014) MATLAB (Version R2014b) [Computer Program]. Available at http://www.mathworks.com/products/MATLAB/ (Accessed 22 July 2015).

[3] Newville, Matt (2014) Wxmplot (Version 0.9.14) [Computer Program]. Available at https://github.com/newville/wxmplot (Accessed 22 July 2015).

[4] Rokem, Ariel. (2015) *Python-MATLAB-Bridge*. [Computer Program]. Available at https://github.com/arokem/python-MATLAB-bridge (Accessed 22 July 2015).