OSG Final Assignment: Ground-Based Gravitational Wave Detection on the Grid

Ryan P. Fisher

August 30, 2010

1 Problem Introduction

Our research group at Penn State is conducting research on gravitational wave (GW) detection with both ground-based detectors and pulsar timing arrays. Our research group has been developing a data analysis pipeline that is designed to analyze very large data sets taken over the period of years to search for and characterize relatively short bursts of gravitational radiation, on the order of a few seconds for ground-based detectors. Due to the computational demands of the data analysis we are using, each few seconds of data can take at least several minutes and up to one to one to two hours to analyze. Thus, we will absolutely have to use a method of high-throughput or distributed computing for our research.

1.1 A Brief Introduction to GWs

I am a member of the LIGO-Virgo Collaboration, and specifically a member of the GW Burst search group of the Laser Interferometer Gravitational Wave Observatory (LIGO) project. This project is designed to detect ripples in space-time known as gravitational waves that are generated by large moving masses throughout the universe. GWs are not absorbed or reflected by ordinary matter but they do produce very small displacements on matter and the effect we are looking for is a displacement smaller than $\frac{1}{1000}$ the diameter of a proton. The key to achieving a measurement of such a tiny displacement lies in a combination of an excellent detector design and very careful discrimination of noise from gravitational signals. Using the current generation of ground based interferometeric detectors, no detections of GWs have yet been made, and a new generation of detectors with ten times better sensitivity is planned to be built and operational within the next decade.

1.2 Current Data Analysis

The LIGO project already uses a grid computing infrastructure to distribute data and allow the submission of analysis jobs on these data sets. There are several different data analysis projects that work within this structure currently, and search for GW signals from differing source types ranging from supernovae and gamma-ray bursts to combinations of neutron stars and black holes that are rapidly orbiting around one another in binary pairs. Our research group is developing an analysis aimed at reconstructing the waveform of a signal once a detection is made. To reconstruct, we require the data from several detectors, a model of the noise appropriate for the observations, and a model of the full response of the detector network (multiple instruments) to an incoming GW. Currently, the most computationally intensive and conceptually challenging aspect of this analysis involves accurately modeling the detector noise and then accommodating the noise model within the waveform reconstruction. Additionally, a full analysis will pair a detection statistic that indicates the likelihood of a detection in a given segment of time with the waveform reconstruction. Although these portions of the analysis are not yet fully complete, we have begun testing the waveform reconstruction on simulated data with white noise using local workstations and the LIGO grid infrastructure, and I will discuss some of the challenges and solutions we would expect to face for a more general distribution over the OSG in Section 3.

2 Estimated Resource Requirements

To estimate the resources needed for future analyses, a few simple tests of a fictional five-detector network with false signal data and white noise were run through the waveform reconstruction portion of our analysis. This code was run using an interactive Matlab session an an estimate of the memory usage was recorded during a sample run. The overhead of running Matlab alone required 1098 MB of virtual space and 163 MB of resident memory space. When the full waveform reconstruction is running, the resident memory increases by only 4 MB. Finally, we currently use 8 cores on our workstation computers through distributed Matlab tools, and the total cpu usage approaches the equivalent of 100% on 6.5 full cores across the system for this test, which only lasts approximately 10 seconds.

For disk usage, ignoring the Matlab runtime libraries (explained below in Section 3.2), the raw input data files for actual LIGO data with a minimal number of channels are only 9.8 MB per detector. These frame files are much larger than the actual data that will be used within the analysis, as each contains 256 seconds of fully sampled, 16 KHz data. A simulated data set within Matlab, in contrast, uses only 343.9 KB of space for 5 detectors with 2 seconds of data taken at 4096 Hz. The associated structure of information including the simulated network response and noise estimation is 1.1 MB.

3 Considerations for OSG Implementation

3.1 General HTC Approach

To implement a general data analysis of months' worth of data across a grid resource, the data segments would be split up across compute resources and the same analysis job would be run for each segment. There are some special concerns to address with LIGO data and security that tie well with the security information covered at the OSG Summer School. The LIGO data is hosted on the LIGO grid, and access to this grid is obtained through an x509 certificate. To access the data, a grid proxy can be generated and then sent to the system or systems that will request the data from the LIGO servers. Thanks to the portability of the proxy certificates, the individual compute nodes could each use a passed grid proxy to connect to the data grid and retrieve the individual data segments using a simple globus-url-copy.

The rest of the workflow for this type of analysis would be straightforward. Each job would only need to generate a relatively short html report to summarize its results and these would then be sent back to the calling computer or sent to a directory on a web server.

3.2 Turning the Analysis into Jobs/Distributing Matlab

With our current testing on the LIGO grid computing resources, we have learned quite a bit about what would be required to transition our analysis to an open grid. One of the first major problems we had to address was that our analysis is written in Matlab and that we currently run it interactively. To run our code on a grid or cluster will require either that each node has a Matlab license and libraries available to it or that the code is compiled and packaged with the Matlab runtime libraries. This will place a restriction on the compute resources used, as the compiled code will only be able to run on the same architecture as the system it is compiled on. In a hypothetical HTC deployment, we would be able to address this by restricting the resources we request within the condor submission scripts to match the appropriate system architecture. Additionally, if the currently implemented distributed Matlab solution will work when spread across a grid, then the systems we run on would ideally be restricted to those with multiple cores available per compute node so that the individual jobs still run in a relatively fast time frame.

To distribute an analysis across the a grid such as the OSG, we will also need to include the self-extracting Matlab libraries with the jobs that are sent out to different compute nodes, have each extract the libraries into a temporary directory, and then finally ensure the local environment points to the necessary libraries at run time. This should be possible by executing a wrapper script for each of the individual jobs to be run, but the drawback of this solution is that the installation and library access can take significant time and will require additional disk consumption, although only temporarily. We have not tested this type of solution yet, so the additional resources necessary will need to be assessed in the future before the appropriate compute node restrictions can be determined.