AGTC

genetic tax. consultants

Requirements

Christian Buskirk
Peter Bellagh
Chris Blazer
Jorden Kreps
Curtis Rose
Faculty Mentor/Project Sponsor Viacheslav “Slava” Fofanov, PhD

1. Glossary

Computing Cluster - A set of loosely or tightly connected computers that work together so that they can viewed as a single system or “supercomputer”.

Configuration File - A file that contains data about a specific user, program, computer, or other file. They are generally read at startup by the operating system and other applications in order to customize the environment for the user.

FASTA - The standard text-based file format used in bioinformatics, where each nucleotide or amino acid is represented using an alphabet of single characters.

FASTQ - A merging of the FASTA format with the read quality data given by the sequencing machine, both still using single characters per base.

GenBank - Genetic sequence database, an annotated collection of all publically available DNA sequences.

Luigi - A Python module that helps build complex pipelines for batch jobs. It handles dependency resolution, workflow management, and visualizations. This is the pipeline management tool that we have selected for this project.

Package manager - A collection of software tools that automates the process of installing, upgrading, configuring, and removing computer programs for a computer’s operating system in a consistent manner.

Pipeline - A pipeline consists of a chain of processing elements arranged so that the output of each element is the input of the next.

Read Set - Genetic information pulled from a biological sample, produced by High Throughput Sequencing Platform(s). Here this is data in FASTQ format, produced by Illumina brand machines (http://www.illumina.com/). This will serve as the primary set of ‘queries’ for metagenomic analysis that is at the core of this project.

Reference Database - A database that holds a large number of genetically classified organisms. Read sets are queried against these to identify organisms in biological samples. This will serve as the primary ‘subject’ set for metagenomic analysis.

Scheduling System - Software that assigns processes to resources on high performance computing clusters, such as time (known as walltime), RAM, and CPU’s. The software can terminate processes that use too much of any given resource.

SLURM - An open source cluster management and job scheduling system. SLURM allocates resources to users interested in using a computing cluster, so that many parties can have access to its cores and computing power.

State System - A system/program that can be in one of a finite number of states. The system/program can transition from one state to another. In our case, the system can transition, linearly, from one state to another while saving the output of those states into files.

Taxonomic Classification - The science of defining groups of organisms based on shared characteristics.

Walltime - The amount of time allocated to a process by a scheduler in which the process must be completed. It the process does not finish before the walltime is reached, the process will be terminated.

2. Introduction

2.1. Who we are

We, the AGTC Genetic Taxonomic Consultants capstone team, are a team of undergraduate students at Northern Arizona University tasked with creating a pipeline management system to handle the massive amount of data generated and analyzed during the taxonomic classification of metagenomic samples. We are:

- Christian Buskirk - Team Lead
- Peter Bellagh - Administrative Assistant
- Chris Blazer - Chief Communication Officer
- Jorden Kreps - Computing Cluster Specialist
- Curtis Rose - Luigi Specialist

2.2. Our Client

The Fofanov Bioinformatics Lab is part of the School of Informatics, Computing, and Cyber Systems at Northern Arizona University. The lab has a focus on the identification of microbes and other life forms in a given sample through the use of High Throughput Sequencing technologies that allow for the determination of all genomes present in a sample simultaneously instead of checking for each potential pathogen individually. This process generates a tremendous amount of data per sample and the problems inherent in the processing of large amounts of data led to them contacting us through Dr. Viacheslav “Slava” Fofanov and the Northern Arizona University Capstone program in search of a potential solution.

2.3. Big Data

To provide a bigger picture of the world that the Fofanov lab plays a part in and just why it is important that they be able to handle the large amounts of data their research generates, it is first important to understand just what Bioinformatics is and what it provides for the world. Bioinformatics is the science of collecting and analyzing complex biological data. Recent advances in Genomics, particularly in the area of High Throughput Sequencing, have produced machines capable of producing billions of bases (characters) of genetic code in a matter of days (see Figure 2.1). This is done by the leveraging of computer power to process the tremendous amounts of genetic data that is present in even a smaller biological sample. Thus, even a single small sample can contain genetic information well in excess of a terabyte, after which that data must be queried against reference databases of several hundred gigabytes in order to match the individual genetic strings present in the sample with their matching references for identification, and correctly classify the sequence. This process can be made manageable mainly through the leveraging of large scale computing clusters to process the terabytes of data present at any given point in time. Through this analysis, newer fields, such as pathogen tracking, are able to make greater progress than ever before, allowing for a level of tracking specific variants of diseases that was previously not possible.

Figure 2.1: Illumina MiSeq, NextSeq, and HiSeq Sequencing Systems.

2.4. Leveraging More Computing Power

A common approach in dealing with these large data sets, similar to what is found in the world of bioinformatics, is to leverage larger computing clusters to aid in processing the terabytes of data that need to be handled. These computers can have hundreds of individual cores present, with several terabytes of RAM available. However these clusters come with another issue of their own: that of scheduling. Due to the large demand for the power of these computing clusters to be applied to many different potential research topics, advanced scheduling systems are utilized to run smaller tasks as space becomes available. Each task is allocated a certain number of cores, amount of memory, and amount of walltime, after which the process will be dumped from memory and another begins. Any research that is done by our client in this computing cluster context, therefore, must be able to address the issues caused by dealing with the scheduling program itself as well as dealing with any issues created by limitations such as wall time and the potential losses that they may cause if violated.

3. The Problem

3.1. Client's Problem

There are several challenges in dealing with genomic data sets and tables in the terabyte range. These issues are:

3.1.1. The client's pipeline needs to be able to break larger tasks down into smaller tasks that can each be run separately. When the basic lookup task as a whole can take up to three weeks of real time, even when run on a computing cluster, it becomes important to be able to easily break a large task down into smaller tasks that can each be individually run as computing cluster workspace becomes available for consumption.

3.1.2. The client's pipeline should waste no time by forcing the reprocessing of any previously processed data. Regardless of failure or branching, the system needs to be able to recover from the current step without any additional reprocessing occurring.

3.1.3. The client's system needs to be able to have individual modules within the pipeline updated without necessitating the entire reworking of the pipeline itself. As Bioinformatics is a relatively new field and large-scale advances in processing techniques are made at a relatively fast pace, this is an integral problem that any potential solution must face. As such the system also needs to be able to track the dependencies of any new modules, to ensure that they are actually able to obtain the needed data prior to reaching the new module itself.

3.1.4. The client needs to be able to easily view a visualization of the current state of the system, allowing them to easily identify how far any analysis has come, what parts if any have failed, and what parts are still waiting to run.

3.1.5. The current pipeline system is deeply linked with the modules that it contains, requiring any changes to the modules to also require changes to the pipeline manager itself. Any replacement pipeline management system must allow for new users to quickly learn how to utilize the pipeline management system without having to inspect the code directly.

3.1.6. The client's system must continue to be maintainable and easily configurable from one centralized configuration file.

3.1.7. The system must be easily deployable to different physical systems, not just the one currently in use by the client.

3.1.8. The client's system must support branching workflows. The system needs to be able to easily branch to different modules using the output of a previous module.

4. System Requirements

4.1. Functional Requirements

4.1.1. The Configuration File

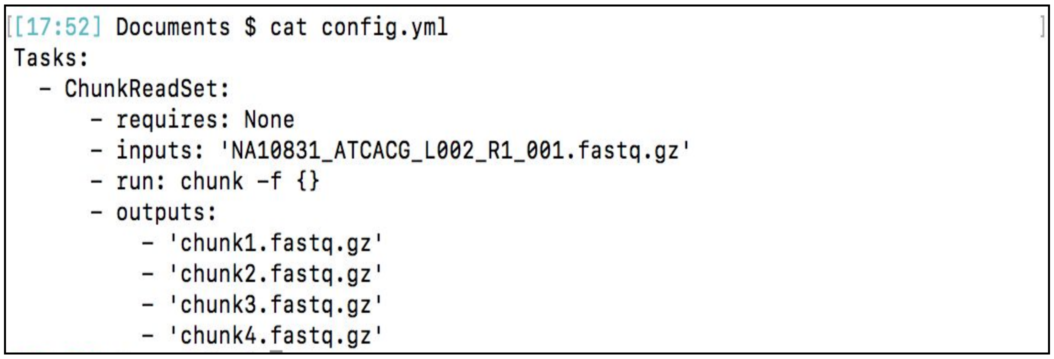
4.1.1.1. There must be a single configuration file that contains settings for all inputs and outputs for a given workflow run by the pipeline manager.

4.1.1.2. The configuration file must be linked to the particular workflow that it is intended for. Each unique branch of the workflow must have its own configuration file.

4.1.1.3. The configuration file must store any file and directory locations that are to be used by the workflow.

4.1.1.4. The configuration file will contain the number of CPU's to be allocated to the workflow.

4.1.1.5. The configuration file will contain the total amount of memory allocated to the workflow.



```
[17:52] Documents $ cat config.yml
Tasks:
  - ChunkReadSet:
      - requires: None
      - inputs: 'NA10831_ATCACG_L002_R1_001.fastq.gz'
      - run: chunk -f {}
      - outputs:
          - 'chunk1.fastq.gz'
          - 'chunk2.fastq.gz'
          - 'chunk3.fastq.gz'
          - 'chunk4.fastq.gz'
```

Figure 3.1: An example configuration in YAML format. One possible configuration format for any potential configuration file.

4.1.2. The State System

4.1.2.1. The pipeline manager must utilize a state system that will allow the workflow to return to any previous state and continue operation from that state in a separate branch of the system.

4.1.2.2. The state system must be able to track if a state is complete.

4.1.2.3. The state system must be able to restart from any previously completed state.

4.1.2.4. The state system must track which version of the pipeline was used to create a given output file, preventing past output from interfering with future runs.

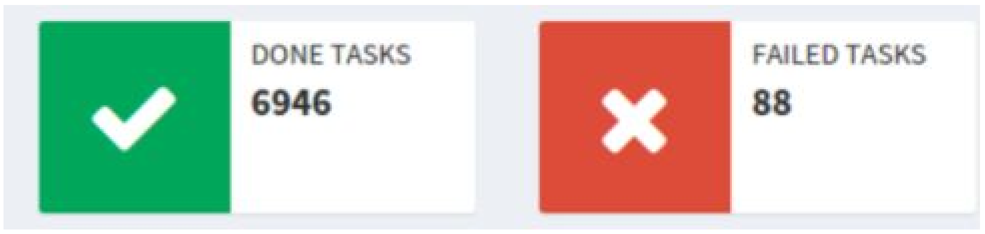


Figure 3.2: An overview of tasks visualization provided by Luigi, allowing the viewer to quickly grasp how many of the tasks in a given pipeline run have succeeded and how many failed.

4.1.3. User Interaction

4.1.3.1. Required user input should be limited to the creation and setup of the matching configuration file prior to the running of an individual workflow without requiring any user interaction during the actual running of the pipeline.

4.1.3.2. The user needs to be able to utilize a graphical user interface to visualize the current status of the workflow both as a whole and on an individual task basis.

4.1.4. File Structure

4.1.4.1. The directories containing the input and output should be able to be moved without causing issue beyond updating the respective configuration files.

4.2. Non-Functional Requirements

4.2.1. Modularity

4.2.1.1. The pipeline manager must be module agnostic. Replacing any module must not require significant pipeline manager changes, and no changes to the modules must be necessary to work with the pipeline.

4.2.2. Resource management

- 4.2.2.1. The pipeline manager must be able to utilize the allocated RAM, CPU's, and time given to it.
- 4.2.2.2. The pipeline manager should require less than 500 MB of memory storage.

4.2.3. Documentation

- 4.2.3.1. The pipeline manager must properly document or provide locations to document:
  - 4.2.3.1.1. The setup procedure for the pipeline manager on a new system.
  - 4.2.3.1.2. The intended behavior of each user parameter.
  - 4.2.3.1.3. The intended behavior of each module.
  - 4.2.3.1.4. The intended dependency linkage.
  - 4.2.3.1.5. The functionality of the pipeline manager as a whole.
- 4.2.3.2. Tutorials must be included for a basic demonstration setup of a new system, including how to modify the workflow by adding or removing modules.

4.3. Environmental Requirements

4.3.1. Integration

- 4.3.1.1. The pipeline manager must be able to integrate with the SLURM scheduling system currently in use by the client.
- 4.3.1.2. The pipeline manager must be able to handle the current modules in use in the client's workflow.

4.3.2. Output

- 4.3.2.1. The output of the pipeline wrapped by the pipeline manager must be able to be written to a predesignated location.

4.3.3. Language Use

- 4.3.3.1. The pipeline manager and all additional parts must be written in Python.

4.3.4. System Deployment

- 4.3.4.1. The system must be able to be deployed on a variety of different hardware and software combinations.
- 4.3.4.2. The distribution system being utilized will be the Python Package Index, PyPi, or pip. This comes pre-existing in almost all python distributions, so will require no workarounds to get to work.
- 4.3.4.3. Documentation and tutorials on how to deploy a new pipeline or update the existing one must be provided with the deployment.

5. A Solution

5.1. A Pre-existing Pipeline Manager

Several pre-existing pipeline managers developed for handling larger amounts of data such as in our client's Bioinformatics workflow already exist. As such, the initial foray into any potential solution comes through the examination and implementation of a pre-existing, off-the-shelf pipeline manager to provide a large amount of functionality and a base for further developments. Through investigation, it quickly became apparent that should a pre-existing pipeline be implemented, several of the client's requirements would already be handled without any further work than what would be required to install and integrate the system. In particular many existing pipeline managers address the following requirements:

5.1.1. Providing a way to only rerun tasks that have updated results from the previous output of the task. In this way they are able to meet requirement 4.1.2 of utilizing a system to not rerun tasks when possible; allowing previously calculated analysis to be instantly utilized without waiting for it to be reprocessed.

5.1.2. Providing distinct separation of the manager from the modules themselves. This meets requirement 3.2.1 that the system must be able to allow for updated modules as time passes. Most pipeline managers also provide for dependency tracking, gaining the ability to identify if a new module is actually able to perform prior to actually encountering it in the workflow.

5.1.3. Finally, many provide an easy way to run tasks separately, in smaller chunks, without necessitating the running of the entire pipeline at once, thus meeting the additional need present in 2.1.1.

5.2. A Specific Pipeline Manager

While all pre-existing pipeline managers that were examined were found to handle several of the client's problems, upon further examination one particular pipeline manager was found to address other potential issues as well; solutions that were lacking in other potential pipeline managers. This pre-existing pipeline manager, called Luigi and developed originally by the company Spotify, also provides the following solutions for the client's problems:

5.2.1. Luigi contains easy compatibility with the SLURM scheduling system (requirement 3.3.1.1), allowing for the client to continue their current research on the new system without any significant interruptions or issues arising due to their ability to run their research on the current computing cluster.

5.2.2. Luigi contains an excellent visualization (see Figures 4.1 and 4.2 below) program directly included in the main Luigi workflow (requirement 3.1.3.2). This will meet the client's need to be able to view the state of the system at any given time, allowing them to easily see what parts of the system are still running, what have already completed, and any parts of the system that have failed, as well as what error messages were produced when those parts of the system failed.

5.2.3. Lastly Luigi also provides a strong, open-source codebase to serve as a backbone for our client's future system, ensuring that should new issues become available they will be able to draw directly on the help of other, open-source programmers to potentially address the issue.

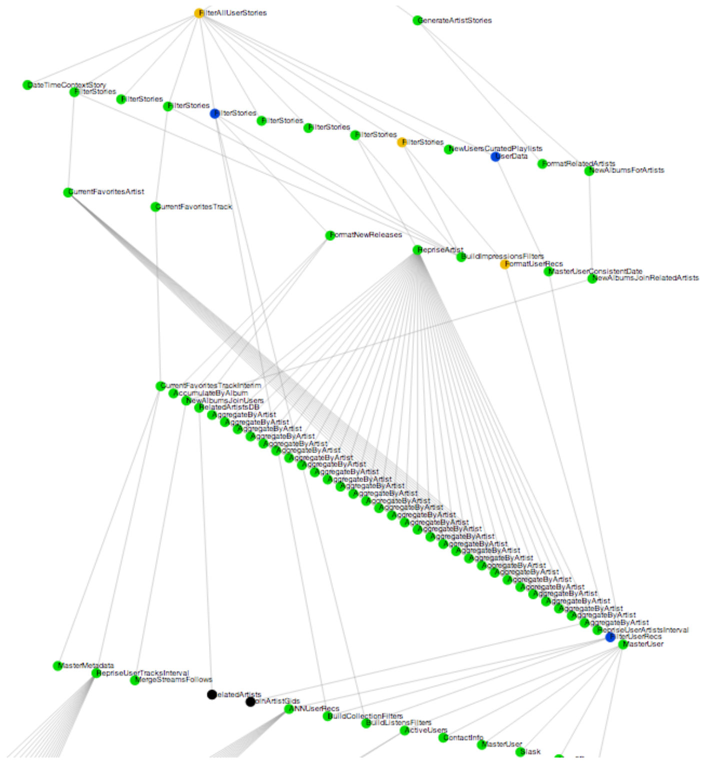
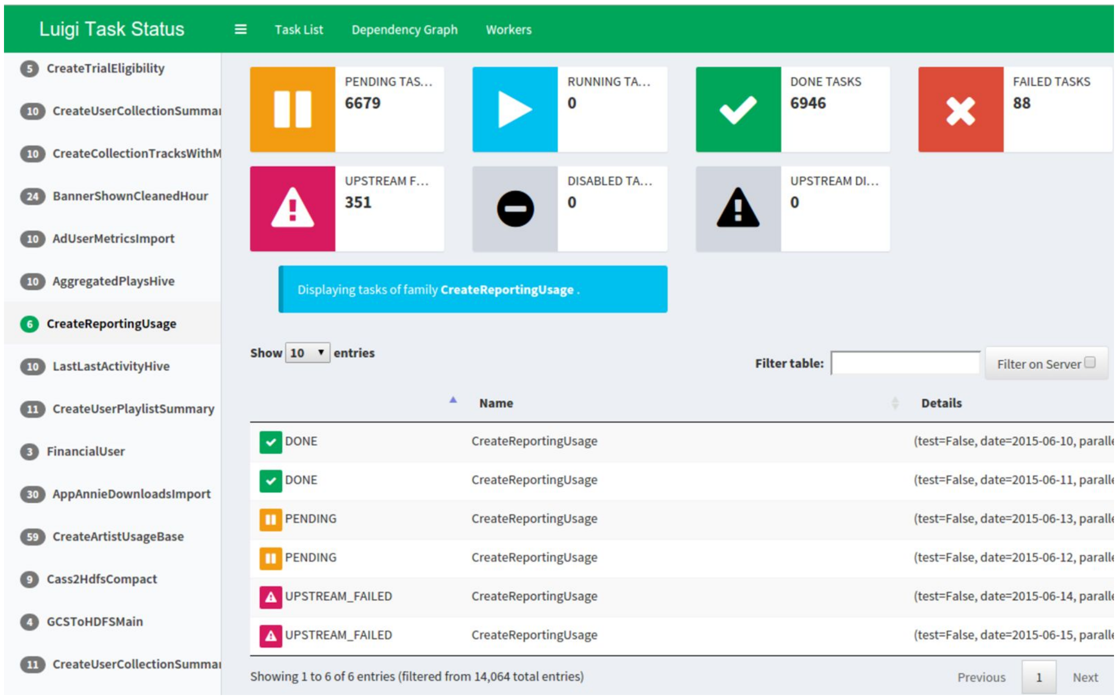


Figure 4.1: A dependency graph of a large scale pipeline in Luigi, showing how Luigi can manage to visualize complex pipeline workflows.



5 CreateTrialEligibility
10 CreateUserCollectionSummai
10 CreateCollectionTracksWithM
24 BannerShownCleanedHour
10 AdUserMetricsImport
10 AggregatedPlaysHive
6 CreateReportingUsage
10 LastLastActivityHive
11 CreateUserPlaylistSummary
3 FinancialUser
30 AppAnnieDownloadsImport
59 CreateArtistUsageBase
9 Cass2HdfsCompact
4 GCSToHDFSMain
11 CreateUserCollectionSummai

PENDING TAS... 6679
RUNNING TA... 0
DONE TASKS 6946
FAILED TASKS 88
UPSTREAM F... 351
DISABLED TA... 0
UPSTREAM DI... 0

Displaying tasks of family **CreateReportingUsage** .

Show 10 entries

Filter table: Filter on Server

	Name	Details
✓ DONE	CreateReportingUsage	(test=False, date=2015-06-10, paralle
✓ DONE	CreateReportingUsage	(test=False, date=2015-06-11, paralle
PENDING	CreateReportingUsage	(test=False, date=2015-06-13, paralle
PENDING	CreateReportingUsage	(test=False, date=2015-06-12, paralle
UPSTREAM_FAILED	CreateReportingUsage	(test=False, date=2015-06-14, paralle
UPSTREAM_FAILED	CreateReportingUsage	(test=False, date=2015-06-15, paralle

Showing 1 to 6 of 6 entries (filtered from 14,064 total entries)

Previous 1 Next

Figure 4.2: The Luigi Dashboard, which can be used to view a large variety of information on both the current run of the wrapped pipeline and information about the pipeline directly.

5.3. Further Work

While Luigi provides solutions to many of the client's problems, there are still a few issues that need to be addressed outside of the bounds of the initial Luigi installation. These issues are those of creating an easy way for less technically proficient people to utilize the system and continue to keep the system maintained as time continues to pass beyond this particular development phase. We propose a twofold solution to address these issues separately, allowing for the remaining client problems to be managed beyond the initial scope of Luigi.

5.3.1. The first step is to separate out as much as possible of the input arguments for the pipeline itself, as well as the settings of the pipeline manager, into a separate configuration file (requirement 3.1.1). This will meet the client's problem of allowing less technical users to be able to edit only a single configuration file, which will be checked to be sufficient prior to the actual pipeline running, rather than requiring them to edit the code of the pipeline itself in order to adjust the desired variables.

5.3.2. Secondly a variety of readme's, other tutorials, and documentation should be created in order to allow for future users and maintainers to be able to quickly grasp how exactly the system works without needing to spend large amounts of time manually learning (requirement 3.2.3). This should meet the client’s issue of being able to continue to maintain the system even after this phase of development has ended.

As a final step we additionally hope to provide easy access to any changes that we implement over the basic installation of Luigi through a basic package manager, allowing them to be easily rolled out and implemented on any future system that the client may move his research onto.

6. Threats and Mitigation Strategies

6.1. New Module Replacement

One threat that faces the implementation of this pipeline manager is a lack of knowledge in how to replace modules within the wrapped pipeline. In order to address this threat, we intend to create ample documentation and tutorials about how to set up and replace modules within an existing pipeline. This should allow new workers with the pipeline manager to be able to quickly grasp the steps required to set up and replace modules. In addition to test the efficacy of the tutorials a presentation will be given to students in the Fofanov lab to ensure that the given documentation is sufficient.

6.2. Future failure of Luigi and SLURM integration

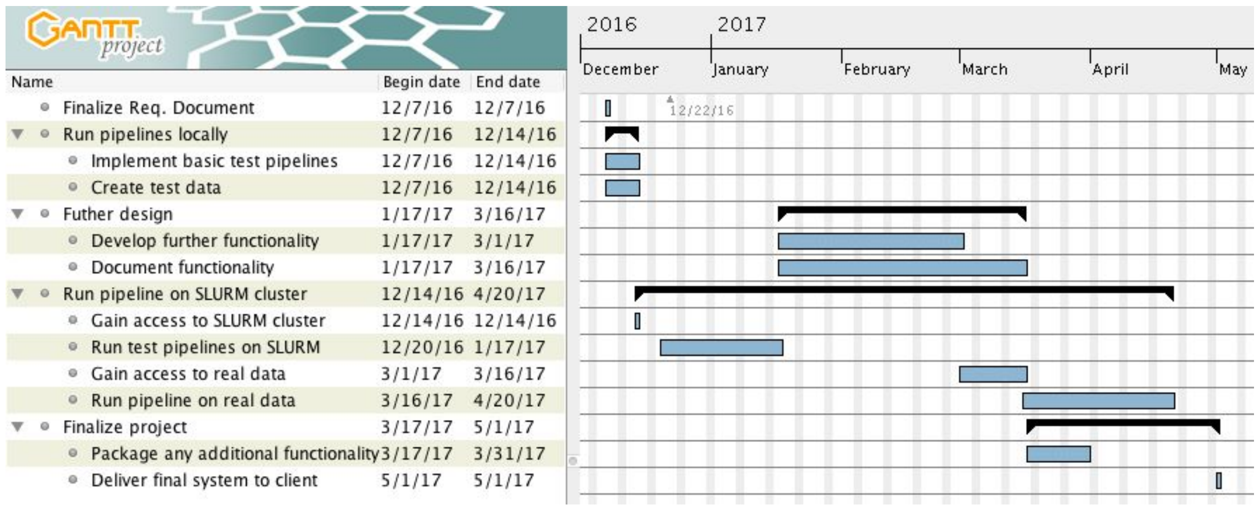
Currently Luigi and SLURM are designed to be able to integrate closely while working together to run pipelines. Should this change, the ability for the pipeline management tool to be used will fail completely. There is no mitigation available for this particular threat and, should it occur, the design choice of the pipeline manager will need to be revisited.

6.3. Failure of the Modules

Should the modules themselves fail then the pipeline manager will be unable to provide functional output. This threat is not one that is mainly able to be mitigated directly through the pipeline manager, and instead should be addressed through mitigation strategies focused around the modules directly. What is able to be done in terms of mitigation through the pipeline manager is to work to capture all available error messages presented by both the modules and the SLURM scheduler itself. This will allow the identification of the reason why a particular run of the pipeline has failed, thus avoiding potential future failures on reruns of that particular pipeline.

7. Project Plan

A projective schedule for the current development plan through the end of the pipeline manager's development may be found in Figure 6.1 below.



Name	Begin date	End date
Finalize Req. Document	12/7/16	12/7/16
Run pipelines locally	12/7/16	12/14/16
Implement basic test pipelines	12/7/16	12/14/16
Create test data	12/7/16	12/14/16
Futher design	1/17/17	3/16/17
Develop further functionality	1/17/17	3/1/17
Document functionality	1/17/17	3/16/17
Run pipeline on SLURM cluster	12/14/16	4/20/17
Gain access to SLURM cluster	12/14/16	12/14/16
Run test pipelines on SLURM	12/20/16	1/17/17
Gain access to real data	3/1/17	3/16/17
Run pipeline on real data	3/16/17	4/20/17
Finalize project	3/17/17	5/1/17
Package any additional functionality	3/17/17	3/31/17
Deliver final system to client	5/1/17	5/1/17

Figure 6.1: A projective project schedule for the remaining development within the upcoming months.

8. Conclusion

Bioinformatics is a new field of research that is now allowing new ways of tracking pathogens, addressing diseases, and other advances based in the world of genetics. In the process though, it generates and requires the processing of terabytes of data, amounts that are only now becoming doable through the aid of larger supercomputing clusters. With this project we hope to allow our client, the Fofanov lab of Bioinformatics at Northern Arizona University, continue to make strides into these fields that can help advances across the country.

By aiding them in the implementation of a pipeline manager to ensure that data is not needed to be reprocessed or lost in the event of a failure, requirements change, or other issue, we hope to enable the saving of weeks of previous research time that could then be redirected at even further advances. We further hope to turn this progress outward, potentially allowing other labs to follow in the steps of our client through our setup and the ease of incorporating and rolling it out, thus saving other labs research time as well that they can turn to advancements in all areas of human and animal life through the field of Bioinformatics.