Section I: Introduction

This document serves as a brief description of the ongoing project: Eusocial Cluster Utility. The GitHub repository for this project may be be found at:

https://github.com/AntonAlbertovich/Eusocial-Cluster-Utility

Eusocial Cluster Utility, otherwise referenced as ECU, has been developed by Anton A Rakos and Sarah K Dominy at Texas Tech University for the purpose of providing Beowulf Cluster Computer users with an easy to install and configure interface for managing machines and distributing task. The need for this utility arises from the lack of such a utility currently being available. As of July of 2019 ECU has incorporated Answer Set Programming as a method for quickly scheduling tasks in a complicated system; a description of this problem is provided in section II of this document. The primary focus of this document is to exsplain how ECU utilizes answer set programming to solve task scheduling problems.

For the sake of providing a better context of how ECU uses answer set program a completely hypothetical cluster network and task list have been provided. Later examples of this project may seek to use more applicable examples.

Eusociality is defined as the highest level of organization of sociality, and describes the cooperative nature of a given community to perform tasks and conduct internal maintenance. When apply this term to super computing, Eusociality means that a tasks are properly delegated and that all the machines are properly utilized. The scheduling of tasks in a cluster can become very complicated if all the machines are not directly connected to one another, if some machine are not capable of solving some tasks, or if some tasks require the prior completion of other tasks and the transfer of their results.

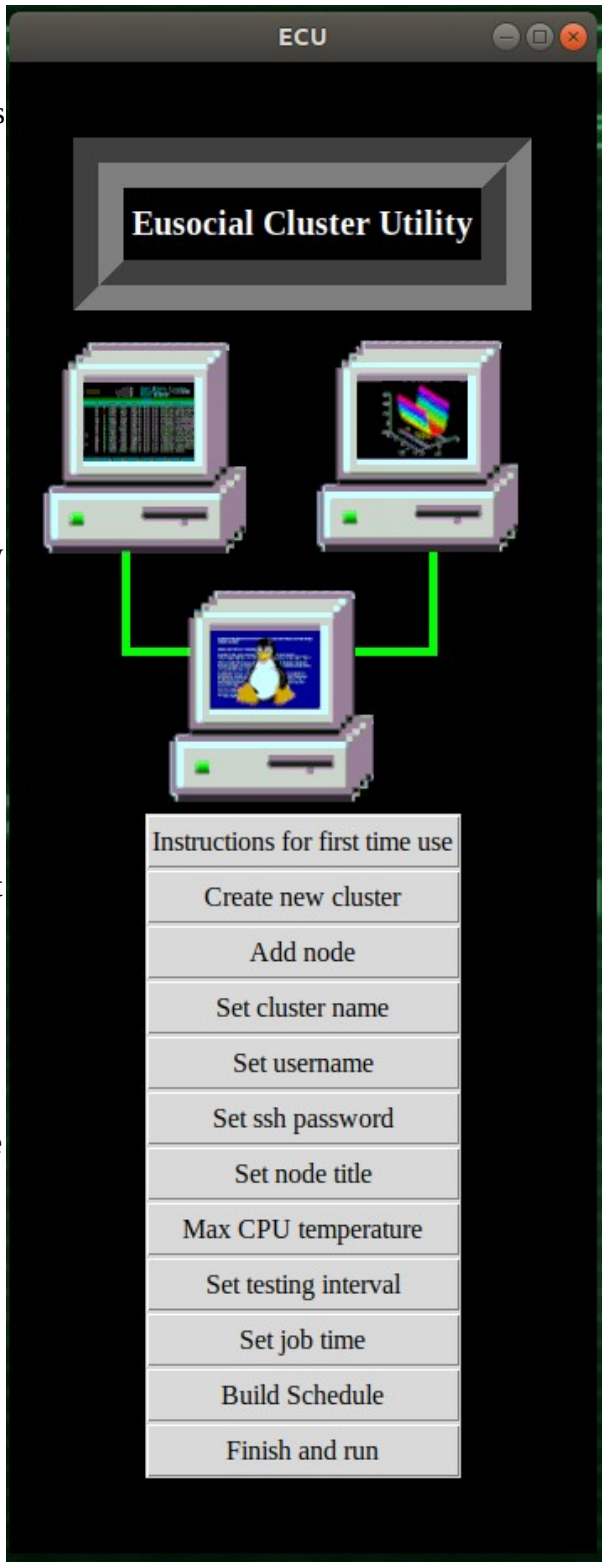

Illustration 1: Eusocial Cluster Utility Main Menu

Section II: A possibly NP-Complete Problem

Let B represent a Beowulf Cluster in where the set M of m many machines are interconnected with the set A of a many authorized ssh connections. The network of B can be represented by a directed graph with m vertices and a edges such that m < a and m > 1. Let H be a subset A0 .. Am of A which represents a set of directed edges from M0 to all members of M.

A tasks is an action which may be transferred from M_i member of M to another member M_j of M so long as there exist an edge from M_i to M_j. A task may only be completed at a member of M if said member meets the proper requirements to complete said task. All tasks begin at M0. No tasks may be completed at M0. Each task will travel once along one member of H to a machine capable of completing said task.

Some tasks t, which is may only be executed on x, may require files produced by the completion a task belonging to the set T. All member of T must be completed prior to the completion of t. If a member of X is not completed on mi then said member must be transferred to x prior to the completion of t.

Given a configuration of some B with the requirements of all tasks corresponding with the capabilities of all members of M, can a plan for transferring and completing tasks be made? If so state said plan.

Eusocial Cluster Utility is capable of solving this problem via modeling the details of tasks and machines in Answer Sets.

Section III: Solving Scheduling with ASP

Assuming that a given configuration of machines is capable of matching the requirements of a given list of programs and said list of programs do not have an impossible dependency hierarchy then answer set programming should be able to present an optimal schedule.

Illustration 2 shows the menu for schedule configuration. For demonstrative purposes I have included several scripts in various languages as well as five different nodes named mac_a through mac_e.

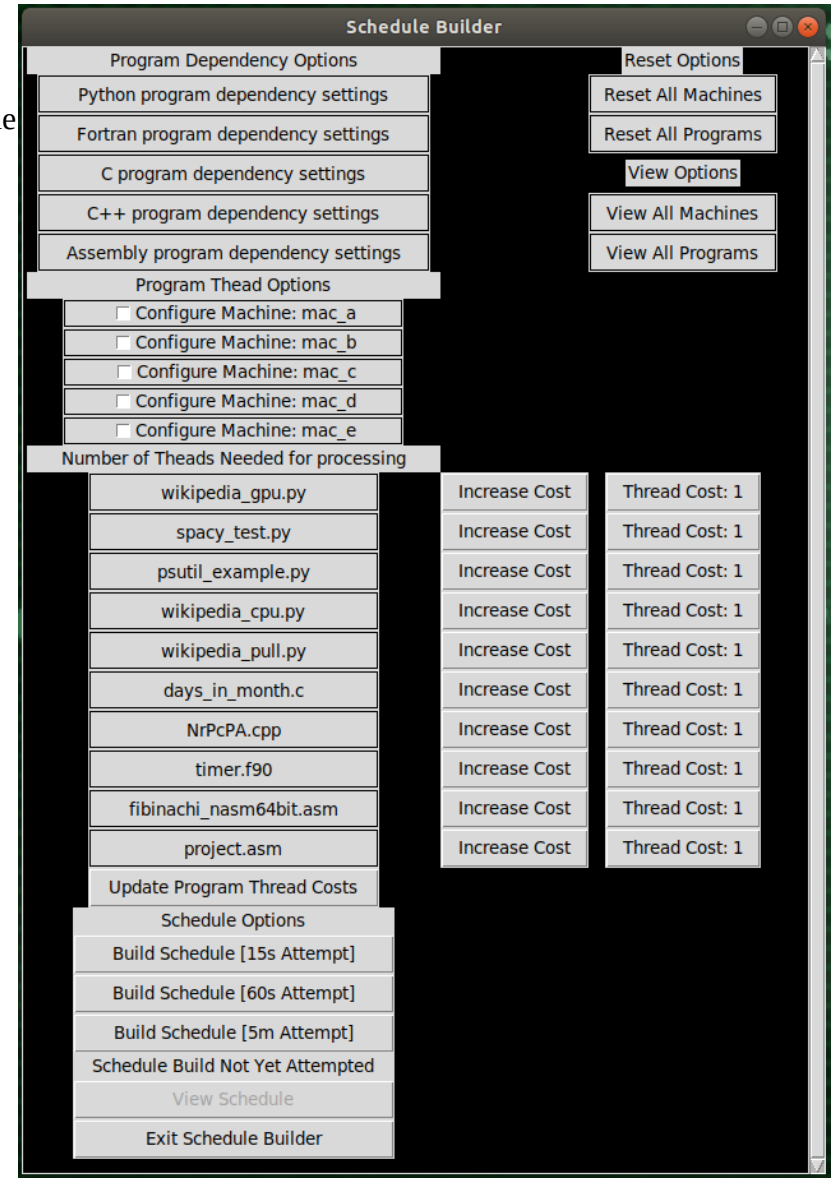

Illustration 2: ECU schedule Builder Menu

Machine Name	MAC address	Machine is connected to	Toolkits on Machine	Machines is avilable at these times	Default execution commands for Python, Fortran, C, C++, and Assembly	Processor Count	Operating System
mac_a	000	mac_b mac_c mac_d mac_e	CUDA spaCy psutil clingo		{python3 } {gfortran } {gcc } g++ {nasm -felf64 }	8	Ubuntu 18.04 [Desktop Edition]
mac_b	001	mac_d mac_a mac_e			{python3 } {gfortran } {gcc } g++ {nasm -felf64 }	4	CentOS 7 [Desktop Edition]
mac_c	002	mac_a mac_b	psutil clingo		{python3 } {gfortran } {gcc } g++ {nasm -felf64 }	4	CentOS 7 [Node/server Edition]
mac_d	004	mac_e mac_c			{python3 } {gfortran } {gcc } g++ {nasm -felf64 }	4	Unlisted Debian based OS
mac_e	005	mac_c mac_b mac_a	clingo		{python3 } {gfortran } {gcc } g++ {nasm -felf64 }	8	Unlisted Red Hat based OS

Illustration 3: A Relation showing aspects of each node in the cluster

Illustration 3 is a cropped view of the relation of the various machines in the cluster and their individual attributes. Here we also see that some machines are not connected to other machines, ECU is intended to eventually be able to help users in distributed long distance networks. Such users may wish to limit what other machines in a given cluster have access to their machine via an SSH communication protocol.

Program Name	OS needed	Program Dependencies	Toolkit Dependencies	Thread Cost
NrPcPA.cpp	N/A	spacy_test.py wikipedia_pull.py timer.f90	CUDA spaCy	1
wikipedia_gpu.py	Ubuntu 18.04 [Desktop Edition]			6
spacy_test.py	Ubuntu 18.04 [Desktop Edition]	wikipedia_pull.py	CUDA spaCy	1
psutil_example.py	N/A			1
wikipedia_cpu.py	N/A			1
fibinachi_nasm64bit.asm	N/A			1
wikipedia_pull.py	N/A			1
days_in_month.c	N/A			1
timer.f90	CentOS 7 [Node/server Edition]			1
project.asm	N/A			1

Illustration 4: A relation of the details pertaining to each program in this example

Illustration 4 shows the various programs which are scheduled to be ran through the example cluster. A program of some interest is the program NrPcPA.cpp which requires that project.asm and fibinachi_nasm64bit.asm be already completed and that whatever files they produced be transferred to the machine executing NrPcPA.cpp. Another program of interest in spacy_test.py which requires that wikipidia_pull.py be already executed and transferred to the machine spacy_test.py is set to run on. Of further interest is the fact thatNrPcPA.cpp requires timer.f90 which must be executed on a machine that is using CentOS & [Node/Server Edition] as an operating system, this presents a challenge and requires that timer.f90 be executed on a machine then moved. Another point of interest is that spacy_test.py must be ran on a machine which already has the CUDA API and the SpaCy Python library installed. All of these tasks are described as not having any multi-threading and thus have a “Thread Cost” of 1. ECU will schedule a machine to run up to three tasks at once, so long as the sum of the Thread Cost for said tasks does not exceed the processor core count for the machine. The only exception is wikipedia_gpu.py which has a listed thread cost of 6.

Section IV Describing a Schedule

As the machines in this example are capable of easily accommodating the tasks as they've been describe, the Answer Set generated a schedule capable of solving this example in three turns. Each "turn" is a period of time where the computation demands placed on each machine are withing what we may assume each machine is capable, likely more development on this area is needed to ensure efficiency.

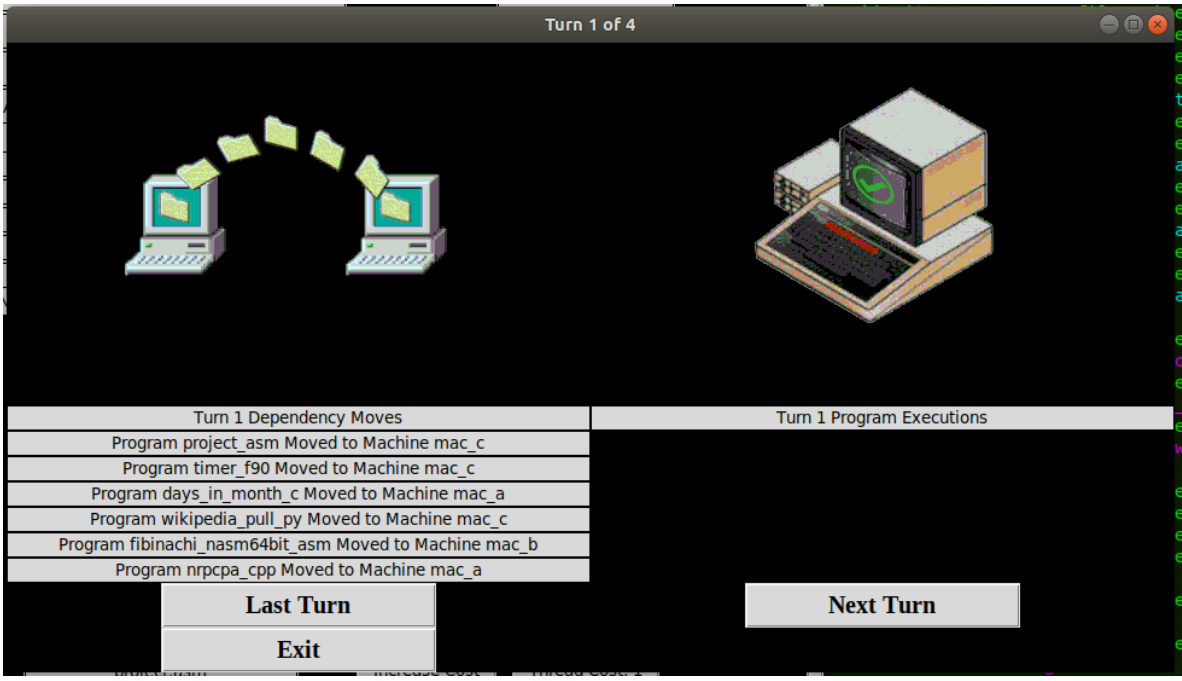

Illustration 5: Turn 1

Illustration 5 shows the first turn of the example cluster's schedule. In this turn many programs are moved to various machines. A program is never scheduled to move to a machine which is not connect to the machine said program currently exists. As answer sets are very capable for solving graph problems, the movement of programs often takes the path of least transfers required to place the programs on the appropriate machines.

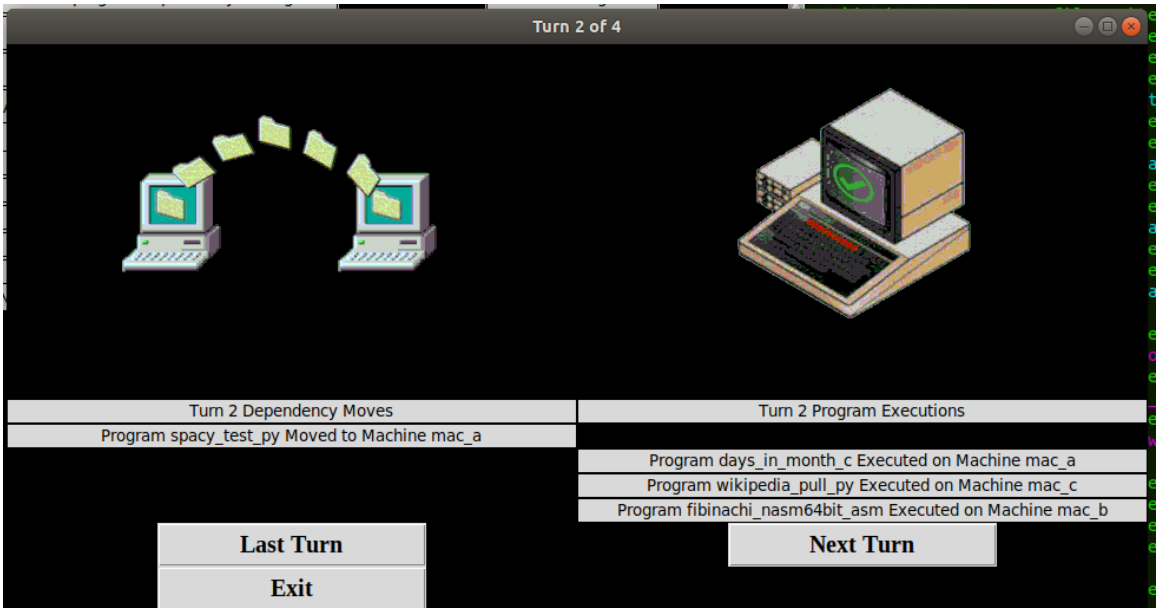

Illustration 6: Turn 2

Illustration 6 shows that several programs are

moved and several are executed. Each turn has both a movement phase and an execution phase. It is worth mentioning that ECU currently inhibits files which are transferred in a given turn to also be executed in said turn. Later additions to ECU may also prevent machine undergoing any sort of file transfer from executing in the same turn as said transfer, given the unpredictable nature of resource costs for file movement.

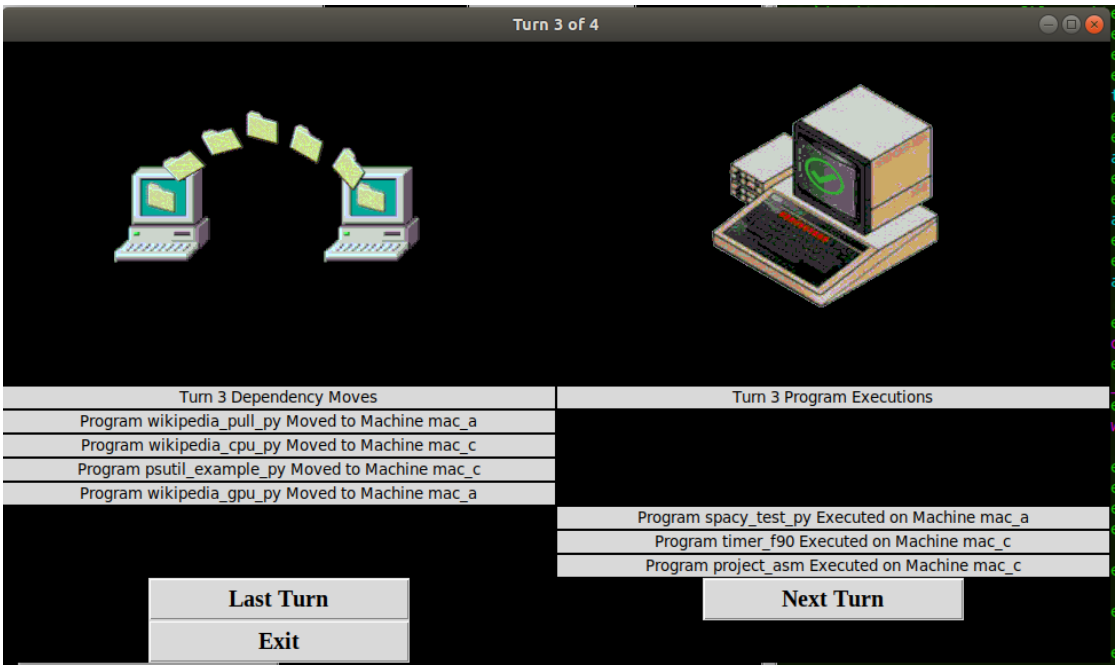

Illustration 7: Turn 3

Illustration 7 shows the final turn of this example. Here we see that the dependencies moved in Turn 3 are being executed on the appropriate machines.

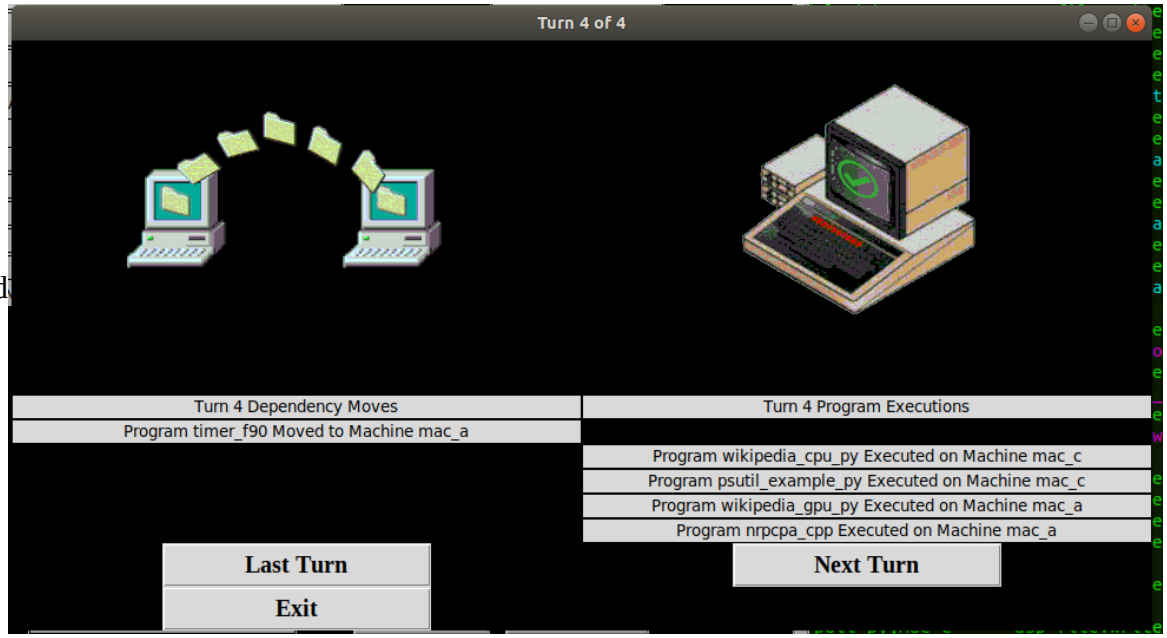

Illustration 8: Turn 4

Illustration 8 shows the final stage of the demonstration. Here we see that only one program is being transferred. Though more complex work was required to achieve this result, the agent will consistently perform as few actions as possible to achieve the end state due to the rule imposed upon said agent.

In conclusion hopefully the reader now holds a better understanding of how ECU utilizes Answer Set Programming to achieve efficient scheduling.