GCC - A Lightweight, Accesible, and User-Friendly Workflow Execution Service

Nathan Loria[1]*

Abstract
A workflow is a cluster of nodes that work together to accomplish an end goal. When executing a workflow, especially one associated with big data from research or other sources, there is often a massive amount of data and time overhead to work around. Because of this, there is a large need for efficient and easy to use software which allows for the execution of workflows on obscure computing resources, which removes the need for massive infrastructure investments by the party that is executing a workflow. Additionally, the demand for efficient task scheduling solutions is ever-increasing. All of these are issues which can be tackled with the proper implementation of a grid computing system. This grid computing approach combined with efficient task scheduling is the focus of our project: GCC. GCC (Gator Computational Cloud) is a lightweight web framework that utilizes a generic task scheduling algorithm to schedule jobs in a cloud environment. This framework intelligently manages dependencies, and takes a multi-threaded execution approach in order to increase efficiency. In order to execute nodes, GCC takes advantage of the Amazon AWS Java API to provision nodes. Once provisioned, the tool completes the execution and transfers any dependencies to their corresponding VM. The goal of the project is to provide a light weight and user friendly environment for workflow execution, while also ensuring a powerful and efficient backend which completes a users workflow with ease. In order to achieve this, some preliminary experimentation has taken place in order to ensure the effectiveness of the tool.

Keywords
Workflow – Grid Computing – Task Scheduling

[1] *Department of Computer Science, Allegheny College, Meadville, PA*
***Corresponding author**: lorian@allegheny.edu

Contents

1 **Introduction** 1
2 **Methods** 2
2.1 AUI 2
Content • Django • Hosting
2.2 CWFP 4
XML Parser • Planner
2.3 NEF 5
2.4 WFPE 5
3 **Experimental Results** 5
4 **Related Work** 5
5 **Conclusion and Future Work** 5
References 5

1. Introduction

As time passes, the world is being driven faster and faster towards a more technocentric and data-centric future. Because of this, there is an ever-increasing need for more efficient, powerful, and usable frameworks for big data management and processing. This phenomenon is known as datafication. It not only raises issues of technical capacity, but also of social equity [2]. Needless to say, it is an issue that is, and will remain, very prominent in almost all aspects of life. With such increase in the overall amount of data that is available in today's world, it can be challenging for people with not a lot of resources or experience with cloud computing to process this data. Even individuals who may have the knowledge to do so, using computational strategies, are often met with the issue of available infrastructure. Computations on large data are almost always extremely resource intensive. This can make it hard for those who do not have access to powerful computing systems to do what they need to do in a reasonable amount of time. This is where the issue of cloud computing and, more specifically, workflow scheduling comes into play.

Already, there can be observed a large shift in focus from local software systems such as older versions of Microsoft's Office (Excel, Access, Word) to cloud-based software such as Google's G-Suite (Docs, Slides, Sheets) for everyday users. This phenomenon is known as software as a service (hereafter referred to as SaaS) and it is becoming very popular in both business and everyday applications. In order to compete with Google, Microsoft released Office 365 which is a cloud-based version of their original suite. This shift was inevitable and now smaller [1] and larger corporations are beginning to convert from previous versions of Office to the newer Office 365 [3] in order to reduce the need for more powerful computational devices. Both Google and Microsoft also offer their own versions of heterogeneous storage systems (Google Drive, Microsoft OneDrive) which are an example of infrastructure as a service (hereafter referred to as IaaS). These two combined principles (SaaS and IaaS) offer a glimpse into the future of computing.

In this paper, we propose a new web-based workflow execution platform called GCC (Gator Computational Cloud). GCC consists of 4 main components: an accessible and easy to use frontend UI (AUI), a workflow planner (CWFP), a node-based execution framework (NEF), and a plan executor (WFPE). Each of these components work together to provide a user with an extremely powerful tool without the need to worry about infrastructure investments or local computer resources. It is for this reason that GCC is an example of a framework that combines both SaaS and IaaS principles. It's web-based frontend combined with a powerful backend where data and nodes can be stored make it a great option for users who may not have the ability to execute their workflows on infrastructure of their own, or pay for the expensive software licenses associated with other localized workflow execution software.

This paper is structured as follows. Section 2 introduces the general structure of the GCC framework and introduces the design of the AUI, as well as the CWFP, NEF, and WFPE algorithms using examples. The preliminary experimental results are discussed in section 3. In section 4, the work related to that done in this project is discussed. Finally, section 5 concludes the paper and discusses future directions and possible areas of improvement to explore.

2. Methods

As previously mentioned, GCC consists of 4 main components: AUI, CWFP, NEF, and WFPE. The first component, AUI (accessible user interface) is a pivotal piece to fostering an effective user friendly environment. The AUI is the source that hosts GCC's web interface. It does so by utilizing the Django Python library and Apache 2 to serve static files. The next component, CWFP (comprehensive workflow planner) is an algorithm that takes in an XML file that is uploaded by a user and translates it into a directed acyclic graph which is used in later execution steps. This algorithm breaks the workflow into levels which allows the WFPE algorithm to do its job properly. The WFPE (workflow plan executor) component is an algorithm that takes in a workflow plan from the CWFP and calls each node's NEF which allows it to run. This is the backbone of the function of the program. The NEF (node execution framework) component is what actually calls the steps to execute a node. This includes the provisioning of instances, the uploading, downloading, and transferring of resources, and the execution of a payload on a node.

```xml
<?xml version="1.0"?>
<workflow>
    <task>
        <id>n1</id>
        <deps></deps>
    </task>
    <task>
        <id>n2</id>
        <deps>n1</deps>
    </task>
    <task>
        <id>n3</id>
        <deps>n1</deps>
    </task>
    <task>
        <id>n4</id>
        <deps>n2,n3</deps>
    </task>
</workflow>
```

Figure 1. A sample workflow specification file (XML)

2.1 AUI

As mentioned before, the AUI component of the project is the component that provides a user with an efficient and accessible interface for operating the GCC service. The implementation is simple yet pivotal to the performance of GCC as a whole. Below are the three aspects of the AUI that enable it to do it's job effectively and efficiently.

2.1.1 Content

The AUI can be seen by travelling to http://141.195.7.181/ on any current web browser that is actively supported and maintained. It consists of a login, signup, about us, and welcome page as well as three menu options entitled "Import a Workflow", "Execute a Workflow", and "AWS Credentials" respectively when on the welcome page. Upon a visit to the website, the user is prompted to input their GCC credentials. If a user does not already have an account with GCC, they are directed to the signup page where they can create an account for the tool. This page contains a lot of validation to ensure that a user provided valid information as well as a unique username. Once a user has created an account with valid credentials, they can then login using the login page. Upon verification of a user's credentials, the login page redirects to the welcome page for a user which gives a brief overview of how the tool works and what it's designed to do. This is the same page that displays the three menu options listed above, and each menu can be accessed at the user's will.

Located on the "Import a Workflow" page is a form which allows the user to upload their workflow source code. For the purpose of this project, a workflow is structured as a specification file (in XML format) and a collection of ZIP archives containing the payload that is to be executed for each node in a workflow. Once these items are uploaded, the tool creates an object for each of them and stores that object in the websites database. In GCC, each ZIP archive correlates to one

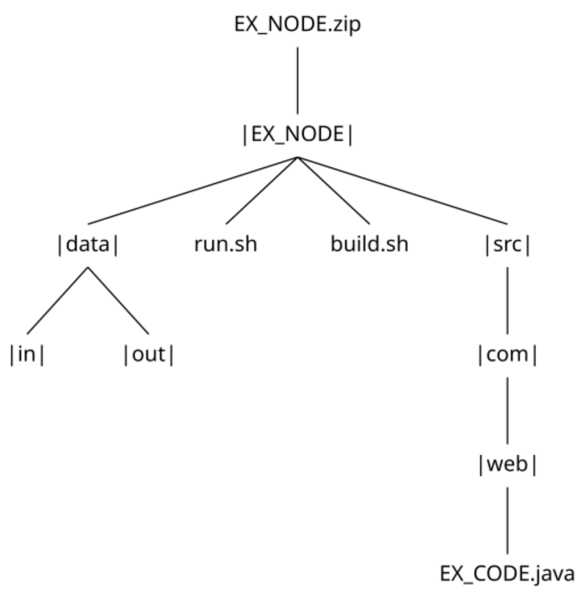

Figure 2. Structure of a sample node archive (ZIP)

node in a workflow. The structure of a workflow specification file can be observed in 1. Additionally, the structure of a node archive file can be observed in 2. In 1, directories are surrounded by vertical lines in order to signify that they are a directory. Any item that is not surrounded by these lines is a file. Please note that for any nodes which require execution-dependent data when they are uploaded, this data should be placed in the "data/in" folder prior to uploading. This figure is an example of a valid node structure starting from the base archive file (EX_NODE.zip) and travelling all the way to the nodes source code (EX_CODE.java). The two scripts (build.sh and run.sh) are used to formalize the execution process and ensure that execution errors do not occur from GCC's side. In 2, the workflow being exemplified is one that contains 4 nodes: n1, n2, n3, and n4. For each node, a list of dependencies is listed. In the case of this example, this workflow would create a diamond shaped when represented graphically as n2 and n3 directly rely on n1, and n4 directly relies on n2 and n3 in order for the execution to function properly. Please note that for each specification file that is uploaded, the name of the workflow object created in the GCC framework will be the name of the XML specification file with the ".xml" extension stripped. The same goes for node object names, except the ".zip" extension is stripped.

Located on the "Execute a Workflow" page is a list of all of the workflow objects associated with a users account. For each object, the website prints out a box containing the name of the object and it's execution status. Workflows that have never been executed will be listed as "dormant" while workflows that are executing or have completed executing will be marked as "executing" or "success" respectively. When a node is tagged as dormant, the user has the option to either execute the workflow or remove it from their account. If the user decides to delete the object, it will be removed from our database and all associated files will be removed from our servers. If a user chooses to execute a workflow object by clicking the button labeled execute, the web server will call the execution server along with a few specific command-line arguments, and the execution server will begin it's process. When a node is done executing, a user can either download the results or delete the workflow. The delete button functions as before however the download button, if selected, returns a ZIP archive containing the logs and results of the execution. This result file is created temporarily on the web server by syncing with the execution server, returned to the users machine, and then removed from the web server to conserve space. If, for some reason, an execution may fail, the object is tagged as "failed" and the user can either download the results in order to view the logs, delete the workflow, or attempt the execution again.

Finally, located on the "AWS Credentials" page is a simple form which allows a user to upload their AWS Access Key, Secret Access Key, and Session Token which are required for GCC to provision cloud instances. These credentials can be accessed either using an AWS Educate account or, more realistically, the AWS CLI by running the "aws sts get-session-token" command for a configured user. This is the simplest of all of the pages however it contains one of the most important functionalities.

2.1.2 Django

In order to generate the website static files, I utilized the Python library Django. Django is a free and open-source web development framework which enables amazingly powerful out of the box performance and a very good amount of customizability. The website takes advantage of a python virtual environment to manage all of its dependencies. Additionally, to keep the website secure, a ".env" file is used on the web server in order to store sensitive information that a hacker could mine from any requests to and from the server.

In order to store information, the website utilizes an sqlite3 database. The database is stored as a local file on the machine and it is where all of the objects and user's information is held. In order to specify the schema of this database, Django utilized what are called models. Models specify a variable name, a data type, and some more metadata about how the data should be treated. This allows users to modify the database schema without the need for advanced knowledge of SQL. In order to ensure security even further, all sensitive information (passwords and AWS credentials) is encrypted prior to going into the database so that, even if somebody were able to view the raw data, it would be extremely unlikely that there would be a data leak. In figure 3, the schema for the database can be seen represented as they are in the Django models file. This well-defined data schema is what allows GCC to function as effectively as it does and it is what allows for objects to be associated with different users. As can be observed in figure 3, both the "AwsAccount" and "Workflow" class

```
class AwsAccount(models.Model):
   user = models.ForeignKey(
                   User,
                   related_name="aws",
                   on_delete=models.CASCADE,
                   default=None
          )
   access_key = EncryptedCharField(
                   default=None,
                   max_length=255,
                   blank=True
          )
   secret_key = EncryptedCharField(
                   default=None,
                   max_length=255,
                   blank=True
          )
   token = EncryptedCharField(
                   default=None,
                   max_length=255,
                   blank=True
          )
class Workflow(models.Model):
   user = models.ForeignKey(
                   User,
                   related_name="wf",
                   on_delete=models.CASCADE,
                   default=None
          )
   name = models.CharField(
                   default=None,
                   max_length=255,
                   blank=True,
                   primary_key=True
          )
   status = models.CharField(
                   default=None,
                   max_length=255,
                   blank=True
          )
class Node(models.Model):
   wf = models.ForeignKey(
                   Workflow,
                   related_name="node",
                   on_delete=models.CASCADE,
                   default=None
          )
   name = models.CharField(
                   default=None,
                   max_length=255,
                   blank=True,
                   primary_key=True
          )
```

Figure 3. Django models utilized by the AUI

contain a foreign key to the "User" class. This is the feature that facilitates the aforementioned association.

The final main feature of Django that will be discussed in this section is the views feature. In order to facilitate a page on a server, Django uses what are called view functions which can be associated with specific URLs. By implementing a view function, you can control which files are served whenever a specific URL is visited. This feature allows the website to process a request, run code that is needed for that request, and return a response in an incredibly intuitive manner. It is in this way that the web server is able to execute a user's workflow object. When the user clicks on the execute button, they are temporarily redirected to the executing view which calls for the execution using a Python subprocess. Once this subprocess is called and is out of the way, the user is then returned back to the "Execute a Workflow" page where they began. It is for this reason that an execution can be run asynchronously so that a user does not have to be logged into the web page in order for their execution to continue. Views provide even further functionality by serving information with each request. This information is what makes it possible to display workflow objects that are associated with a user as well as serve forms that contain the means to receive information.

2.1.3 Hosting

In order to serve the static files generated by the Django source, GCC utilizes an OpenStack instance which is configured with an Apache 2 server. The reason for this is that the instance can be constantly running and serving files, making it available to all at any time of the day. The Apache 2 server is configured to read files from a directory called "gcc_dep" and it is instructed to use the environment which is available in this directory to ensure that all required packages are being utilized. For an added layer of security, all files on the server are access restricted so only sudo users can access these files. The only way to gain sudo privileges is to connect over ssh and that is restricted to only those who have access to the machine's secret key. This added layer of serves to further enforce strict data security practices.

2.2 CWFP

The CWFP (Comprehensive Workflow Planner) is an algorithm, written in java, that intakes a workflow specification file from a user and converts that file into a directed acyclic graph. This graph represents the dependencies between nodes in a workflow. It consists of two main pieces: an XML parser, and a planning method which turns the raw information into the desired directed acyclic graph.

2.2.1 XML Parser

The XML parser is the portion of the CWFP that handles the raw input data. When a workflow is set to be executed, the XML parser intakes the path to a specification file, the name of the user who's requesting an execution to take place, and the name of the workflow which is to be executed. In order to perform the parsing procedure, this method utilizes the "org.w3c.dom" and "javax.xml.parsers" libraries which work together to break down an XML file into different objects which can then be analyzed and have the data stripped out of them. Initially, the document is converted into a "NodeList" object using the method "getElementsByTagName('task')". This method reads in the XML data and creates a list of each object labeled "task" that is within the specification file. An

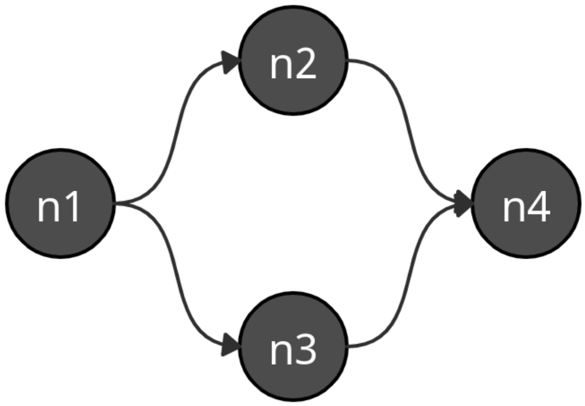

Figure 4. Structure of a sample node archive (ZIP)

example of this is represented in figure 1 where each element consists of an object with the label "tag". After this list is created, it is iterated over using a for loop and the contents of each object are extracted and stored as string variables within the loop. This information is then placed into a new NodeADT object and that node is added to an ArrayList as well as a HashMap which is used to correlate a node's id to its corresponding NodeADT object. This information is stored locally within the CWFP class and it is not returned to the main method since it is not used there.

2.2.2 Planner

As mentioned before, the workflow planner creates a directed acyclic graph data type to represent a workflow object. In order to do this, the planning function utilizes the ArrayList and HashMap that are created by the XML parser. Firstly, two ArrayList objects called "lint" and "compIds" are created, as well as an integer "max" which is initialized to zero. Next, a while loop is created and it runs until it is broken out of using a logical break statement within the loop. Also within this loop, there are two for loops. These for loops are not nested. In the first for loop, all of the nodes in a HashMap "hm" are iterated through. Initially, "hm" is empty so the first for loop does not run on the first iteration of the while loop. If "hm" is populated, "lint" and "compIds" receive the node and node id for each object within "hm" respectively. Once this is done, a logical check happens to see if "lint" is equal to the ArrayList set by the previous section. If this is true, then the while loop is broken out of. If it is false, the second main for loop is invoked. This check is to ensure that when everything is added to the graph that the while loop does not continue to run. If the second main for loop is invoked, meaning the planning has not completed, it iterates through each node in the ArrayList from the XML parser. For each node, there is some validation that occurs to make sure it is valid and exists, and then it is added to the HashMap "hm". This marks the end of an iteration of the outer while loop and, depending on if "hm" is fully populated or not, it will run again. Once outside of this while loop, all levels within "hm" are inserted into a directed acyclic graph and this is returned to the user. Each graph contains a set of nodes, and each node contains a set of edges which allows for an efficient execution once the graph is created. Figure 4 shows the directed acyclic graph that would result from running the workflow specification from figure 1 through the CWFP.

2.3 NEF

2.4 WFPE

3. Experimental Results

4. Related Work

5. Conclusion and Future Work

References

[1] Kim, S. H., Jang, S. Y., and Yang, K. H. Analysis of the determinants of software-as-a-service adoption in small businesses: Risks, benefits, and organizational and environmental factors. *Journal of Small Business Management 55*, 2 (2017), 303–325.

[2] Mejias, U. A., and Couldry, N. Datafication. *Internet Policy Review 8*, 4 (2019).

[3] Skendzic, A., and Kovacic, B. Microsoft office 365-cloud in business environment. In *2012 Proceedings of the 35th International Convention MIPRO* (2012), IEEE, pp. 1434–1439.