CHAPTER 1

INTRODUCTION

1.1. Rationale

Cloud Computing has become one of the most talked about technologies in recent times. Cloud's powers are undoubted in offering various benefits in plentiful areas [1] such as Biotechnology, scientist community… This technology reduces the entrance barrier for new service providers to offer their respective competencies to wide market with a minimum of entry cost and infrastructure requirements. Recently, cloud computing has risen as an ideal solution for scientific applications and is rapidly gaining interest in the scientific community, namely, scientific workflow application. Workflows [2] are loosely-coupled parallel applications [3] that consist of a series of computational tasks connected by data and control-flow expressed as workflow, and they are commonly used to solve problems in many disciplines, such as computer sciences, astronomy, biology…

One of the imperative problems of scientists who are using grid resources for large-scale applications is managing every part of the application manually, such as submission of tasks; waiting for the completion of one task or group of tasks in other to submit the next; submitting hundreds of parallel simulations at the same time. One solution to eliminating human interference and simplifying the management of such applications is via automation of the end-to-end application process using workflows. The above Figure will explain one simple example of workflows

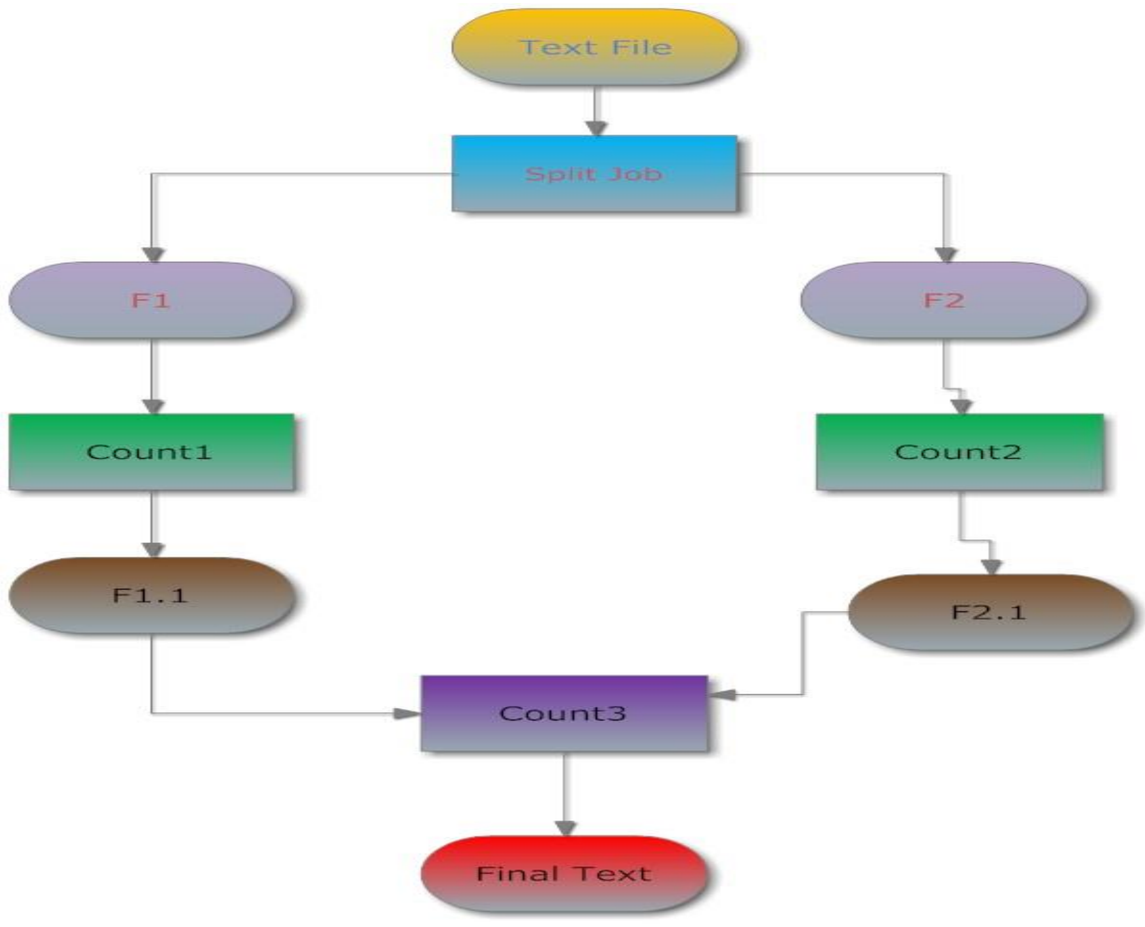


Figure 1- CharacterCount workflow

Assume that there is a problem that requires counting character included in one text file. There are many approaches to solve this problem. Thus, my method is not an ideal solution, and it is just used as a means to explain what the workflow is. This issue is solved following as above figure. At first, the content of the text file was equally split into two different new text files. Next, the two new text files are concurrently counted by two different scripts, Count1 and Count2. The result of this process is two new files, F1.1 and F2.1. At the final stage, the script Count3 takes responsibility in counting the two F1.1 and F2.1 to output a final text, which includes the total words in the First File. The whole process explained above is called workflow, which is very popular in the scientist community. The approach of my paper is to build a system that solves those workflows by taking advantage of Cloud's powers.

In this paper, I am considering; a cloud is a group of machines configured in such a way that an end-user can request any number of virtual machines (VMs) based on their demand. The end-user neither knows nor cares where exactly these VMs are physically located or the configuration of the underlying hardware, so long as they can access their bank of properly configured VMs. The VMs are used as a computing resource to compute their own jobs. All these operations interact via the web interface. This is one of the main reasons that the cloud platform is chosen to replace the grid platform, and this issue will be discussed in detail in the next section. The kind of setup is ideal for applications, namely workflow applications, where a specific hardware configuration is needed, or users need the high computing capacity.

The below figure explains the general model of my work. All the VMs act as worker node, all jobs are executed there. Thus, the utilization of resource can be significantly increased

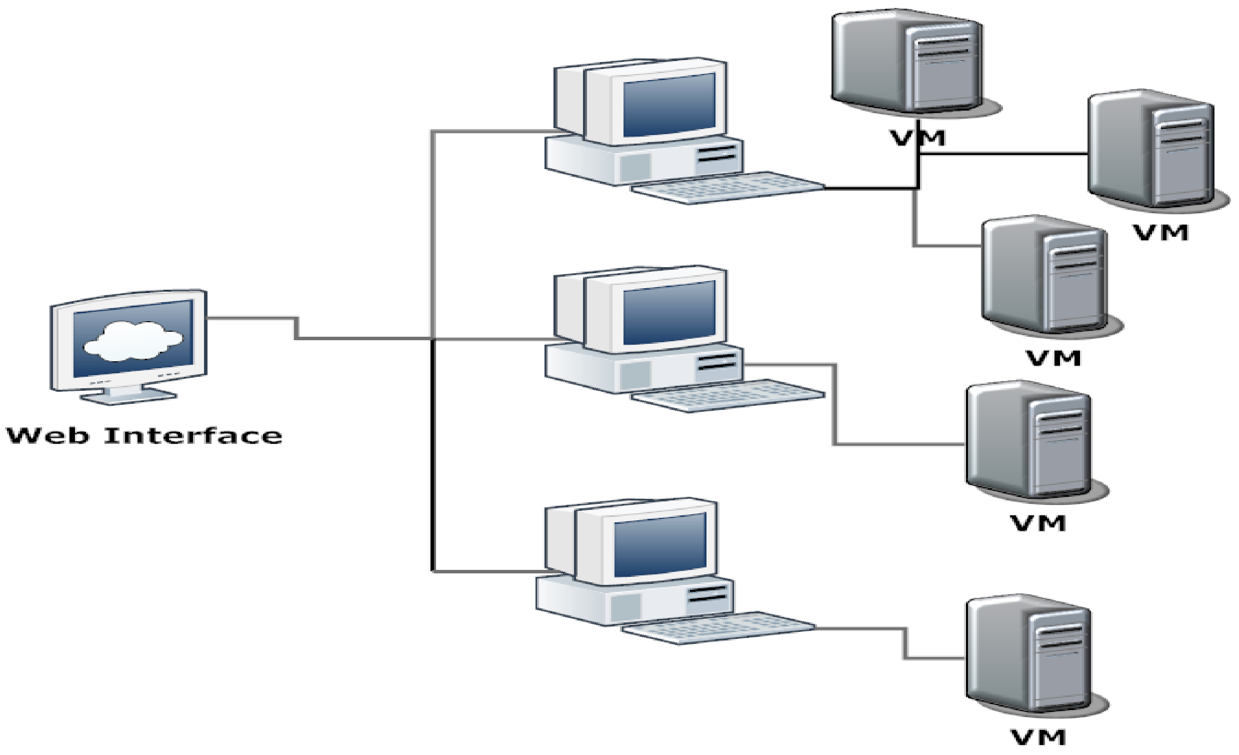


Figure 2 – Sample Computing System Architecture

However, to solve complicated workflows, high demand of computing resources for job processing that require hundreds or even thousands of cores are required. Depending on the computational settings and the problem being solved, this thesis program was taken.

All those above explanations are ideas for us to carry out the goal of this paper, that is to study the implementation of scientific workflow application, namely, Pegasus [4], Condor [5], DAGMan [6] on an open-source cloud platform, EUCALYPTUS [7] [8] [9].

1.2. Thesis Statement

According to the above context, the contributions of this paper are:

- ➢ Understand the architecture of Eucalyptus Cloud, along with its constraints
- ➢ Establish a private cloud using Eucalyptus platform.
- ➢ Technically show how to deploy computing service with Pegasus, Condor on private cloud.
    - Create image including configuring storage and networking.
    - Build, deploy, manage Centos image included required software on cloud.
    - Configure the user environment and perform maintenance on the cloud
    - Provide a fully computing platform-based cloud system.

1.3. Scope

The thesis concentrates on studying the knowledge of Eucalyptus and understands the implementation of workflow applications into cloud platform. The Centos Operation System version 5.5 was chosen to implement Eucalyptus and the rest of this works.

1.4. Limitations

- Due to the complication of the use of workflow applications, this work only intend to expert users.
- Due to the lack of hardware, the power of the platform might be underestimated; this platform is not able to compute some complicated tasks. In another hand, with a large organization with many users, it might be more cost effective for the organization to purchase hardware to create its own private computing cloud platform.
- The whole system was built on an open-source platform; thus, bugs are unavoidable.
- Finally, cloud-computing system is a big challenge for me to develop from the bottom to the top, thus, this thesis can be finished in some certain respects.

1.5. Thesis structure

The report is divided into six chapters. The first chapter, “Introduction”, expresses the perspective of the whole project throughout the definitions; it includes the motivations, limitations, scopes, and the structure of the paper. The next chapter, “Literature Review”, points out the technical characteristics and the specifications of the main elements in this thesis. In this chapter, definition stated in chapter 1 will be illuminated in detail. After that, the main architecture of this project is handled in chapter three. Next, chapter 4 describes the process of implementation of all works that were done in this study, namely, installing Eucalyptus, Pegasus, Condor, DAGMan. The next chapter handles the evaluation of the system.

Finally, Chapter 6 gives a briefly summary of contents and works throughout the thesis program, namely Conclusion. Additionally, future works are also offered in this chapter.

CHAPTER 2

LITERATURE REVIEW

2.1. Grid Computing

In the past, grid computing was knows as an ideal environment to solve a large scientist problem, such as image processing [10], energy resources exploration [11]… by taking advantage of idle resources of various OS-based computer from remote area. There are many definitions in term of Grid Computing, according to wiki [12], "***Grid computing*** *is a term referring to the federation of computer resources from multiple administrative domains to reach a common goal. The* ***grid*** *can be thought of as a distributed system with non-interactive workloads that involve a large number of files. What distinguishes grid computing from conventional high performance computing systems such as cluster computing is that grids tend to be more loosely coupled, heterogeneous, and geographically dispersed*". However, since Cloud Computing has been invented, with its powerful features, the approach about using grid computing in scientist community should be reconsidered. The next section, a brief comparison between Grid and Cloud are taken.

2.2. Compare Grid Computing and Cloud Computing

Although Grid Computing has been used as a best solution in solving parallel problems for many years, yet Cloud Computing with its own powerful characteristics has slowly dominate Grid Computing due to following reasons.

Grid Computing is focused in solving computational problems whereas Cloud Computing has no limitation, Cloud does not only offer computing service but also various types of services, such as:

- **SaaS** (Software as a Service)
- **IaaS** (Infrastructure as a Service)
- **PaaS** (Platform as a Service)
- **CaaS (**Computing as a Service**)**

Grid Computing is possible only with applications that can be parallelized whereas Cloud computing does not have any such restriction.

In Grid, resource allocation is primarily controlled by the contract between the user and allocator. In general speaking, there is a limitation with the dynamic sharing resources availability, in some specific moment; the remote resources may not meet the demand of customer, and furthermore, the integration between nodes, security is also an issue of networked environment. While in Cloud there is no restriction. Cloud computing is a set of virtual servers that work together through the internet or locally. Based on Virtualization Technology [13], VMs can be configured as independent environments from physical resources, in this manner, sharing will be much more flexible, the resource utilization also is increased, thus the requirements of customer will be assured.

2.3. Cloud Computing System

2.3.1 Cloud Computing Overview

Cloud Computing is defined as a pool of virtualized computer resources. Based on this Virtualization the Cloud Computing paradigm allows workloads to be deployed and scaled-out quickly through the rapid provisioning of virtual machines or physical machines. A Cloud Computing platform supports redundant, self-recovering, highly scalable programming models that allow workloads to recover from many inevitable hardware/software failures.

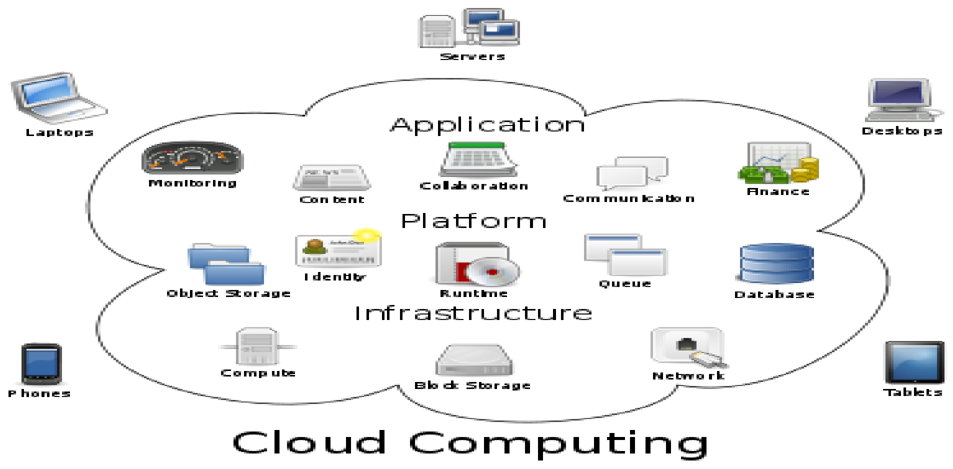


Figure 3 – Cloud Computing

The shift to Cloud computing is driven by economies of scale. The cloud providers take advantage of a relative cost reduction, when they build large data centers. It enables them to buy hardware in enormous quantities, which results as cheap compute power. Cloud computing has a market-oriented business model in which users are charged for their demand. These cloud services include the contribution of compute power, storage and network services. Charging for these resources analogously occurs to conventional utilities in daily life. This concept is called utility computing.

The powerful feature of Cloud Computing is offering elastic services on-demand resources over the Internet. This causes the illusion of an infinitive amount of on-demand computing resources, which allows companies start small and expand their business when customers require more resources. This elasticity eliminates the need for an up-front commitment to a fixed-size long-term investment in hardware, as found in traditional privately owned data centers.

2.3.2 Service Models

Cloud computing can be broadly classified into three ***aaS**, namely:

- Infrastructure-as-a-Service **(IaaS)**
- Platform -as-a-Service **(PaaS)**
- Software-as-a-Service **(SaaS)**

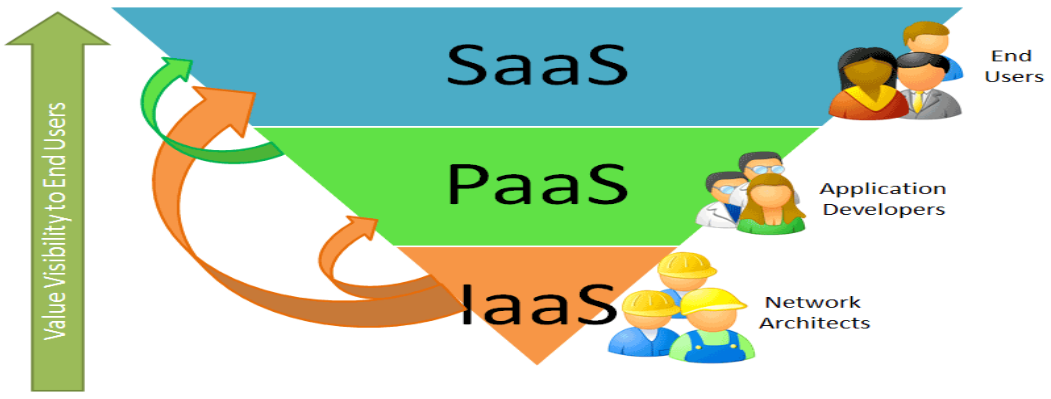


Figure 4 - Cloud Stack

Infrastructure-as-a-Service (IaaS)

This is the base layer of the cloud stack. It serves as a foundation for the other two layers, for their execution. The keyword behind this stack is *Virtualization*. The service provider owns the equipment, such as, servers, routers, hardware-based load-balancing, firewalls, storage, and is responsible for housing, running, and maintaining it. The client typically pays on a per-use basis. So, client saves his money as the payment is only based on how much you use only.

Platform-as-a-Service (PaaS)

PaaS is a category of cloud computing services that provide a computing platform and a solution stack as a service. In the classic layered model of cloud computing, the PaaS layer lies between the SaaS and the IaaS layers.

PaaS a concept is known as Platform as a Service and as Cloud Computing Platform also. PaaS applications are referred as On-Demand, Web-based or as Software as a Service (SaaS) Applications.

Software-as-a-Service (SaaS)

It is on-demand software delivery model in which software and its associated data are hosted centrally on cloud and are typically accessed by users using a client, in other words the user can access the service through the Internet with only a browser installed on his device. Although applications are deployed on cloud, but they are still configurable by the customer. The service provider will take responsible for the management of software, thus there is no need for the user to care about these jobs.

2.3.3 Cloud Deployment Models

- ***Public Cloud***: The cloud infrastructure is owned by an organization selling cloud services to the general public or to a large industry group.

  Examples of Public Cloud:

  - Google App Engine
  - Microsoft Windows Azure
  - IBM Smart Cloud
  - Amazon EC2

- ***Private Cloud:*** The cloud infrastructure is owned or leased by a single organization and is operated solely for that organization.

  Examples of Private Cloud:

  - Eucalyptus
  - Ubuntu Enterprise Cloud - UEC (powered by Eucalyptus)
  - Amazon VPC (Virtual Private Cloud)
  - VMware Cloud Infrastructure Suite
  - Microsoft ECI data center.

- ***Hybrid Cloud:*** The cloud infrastructure is a composition of two or more clouds (internal, community, or public) that remain unique entities but are bound together by standardized or proprietary technology).

  Examples of Hybrid Cloud:

  - Windows Azure (capable of Hybrid Cloud)
  - VMware vCloud (Hybrid Cloud Services)

- ***Community Cloud:*** The cloud infrastructure is shared by several organizations and supports a specific community that has shared concerns (e.g., mission, security requirements, policy, and compliance considerations).

Examples of Community Cloud:

- Google Apps for Government
- Microsoft Government Community Cloud

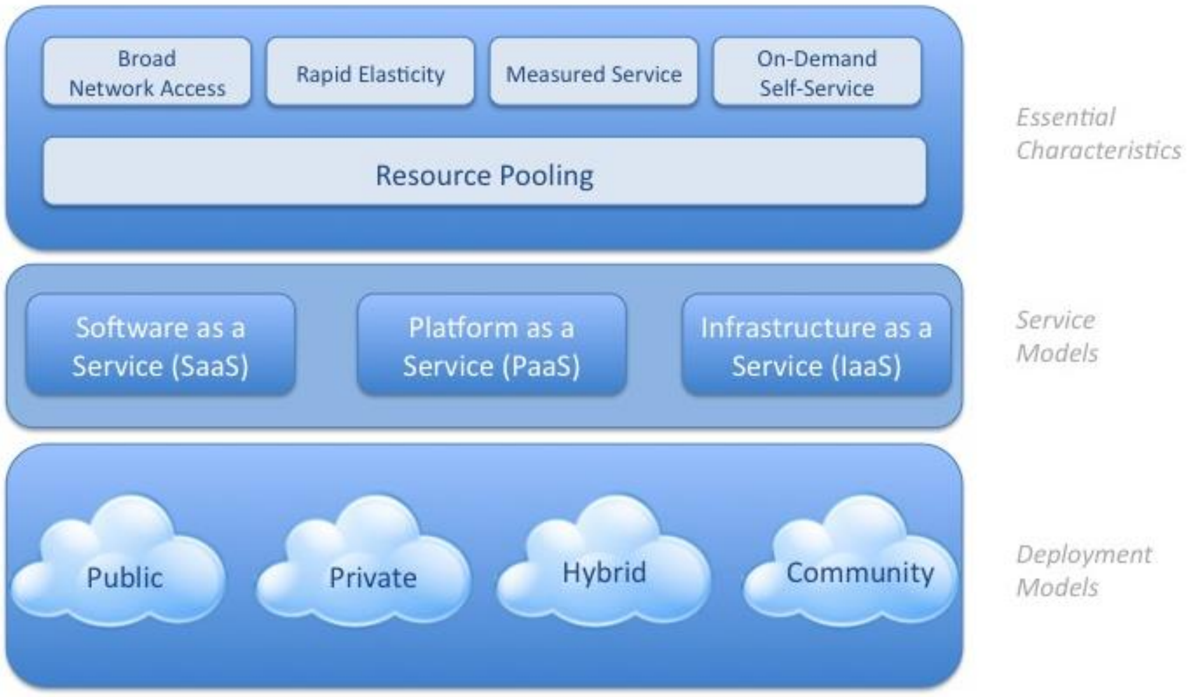


Figure 5 - Visual model of Working Definition of Cloud Computing

2.3.4. Characteristic of Cloud Computing

This section gives an overview of the most important characteristics and associated advantages of cloud computing

On-Demand self-service: individuals can set themselves up without needing anyone's help; Cloud resources are available in an on-demand manner, which means the customer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service's provider. Cloud computing brings about a certain level of agility that enables a fast time to market for a number of new software products, since these can be hosted on cloud resources that are available on-demand.

Ubiquitous network access: The same with another internet-based application, cloud resources are accessible over the Internet, which makes the access to the application hosted on these resources device and location independent.

Location independent resource pooling: processing and storage demands are balanced across a common infrastructure with no particular resource assigned to any individual user. In another words, the computing resources of the provider are pooled to serve multiple consumers that are using a multi-tenant model, with various physical and virtual resources dynamically assigned, depending on consumer demand.

Rapid Elasticity: Cloud computing system can be rapidly and elastically provisioned.

Pay-what-you-use: consumers are charged fees based on their actual usage of a combination of computing power as well as bandwidth use or storage. Therefore, it is important that the resource usage is measured properly.

Virtualization: Virtualization techniques make it possible to run multiple virtual machines on a single physical machine. Cloud computing, for instance, is the virtualization of computer programs through an internet connection rather than installing applications on every physical computer.

Green IT: Cloud computing is obviously a form of a green IT, since it is ecologically friendly to use a data center that increases the utilization rate of hardware, which lowers the total power usage. Furthermore, with virtualization technology, cloud system reduces a large amount of physical hardware.

2.4. Cloud Computing Software platform to implement private cloud computing

The purpose of this part is to examine software platforms to implement private cloud computing. In this dissertation program, the main goal is to deploy a collection of computational resources, included workflow applications, thus, Platform as a Service is selected.

Obviously, there are two approaches to build a private cloud platform, using open source and close source. Open source such as Amazon Elastic Compute Cloud (Amazon EC2) [14] is an ideal tool to deploy a private cloud; however, the users have to pay a lot of money to use the service. Therefore, the second choice, open source softwares, is a suitable method for this undergraduate program.

Open source solutions are generally available in both source code and binary forms on-line and without charge. Communities of users and developers contribute the labor necessary to develop and maintain them. According to Sebastian Rupley (2009) [15], Open-source software has been on the rise at many businesses during the extended economic downturn, and one of the areas where it is starting to offer companies a lot of flexibility and cost savings is in cloud computing. Cloud deployments can save money, free businesses from vendor lock-ins that could really sting over time and offer flexible ways to combine public and private applications.

Eucalyptus, OpenNebula [16] and Nimbus [17] are three major open-source cloud computing software platforms. OpenNebula is geared toward persons interested in the cloud or VM technology as its own end, it is also a suitable solution for anyone that wants to quickly stand up just a few cloud machines. Nimbus looks toward the more cooperative scientific community that might be less interested in the technical internals of the system but has broad customization requirements. Among these softwares, Eucalyptus contains more advantages than the others for the following reasons.

Firstly, Eucalyptus tends to a private organization that wants their own cloud for their own use and wants to protect themselves from user malice and mistakes. Furthermore, according to Wikipedia website, it claimed that Eucalyptus operates with most currently available Linux distributions such as: Ubuntu, Red hat Enterprise Linux, CentOS or OpenSUSE, additionally, Eucalyptus can use a variety of virtualization technologies including VMWare [18], Xen [19], KVM hypervisors [20] to implement the cloud abstractions it supports.

In technology aspect, Eucalyptus was designed from the ground up to be easy to install and as nonintrusive as possible. The software framework is highly modular, with industry standard, language-agnostic communication. Eucalyptus is also unique by providing a virtual network overlay that both isolates network traffic of different users and allows two or more clusters to appear to belong to the same Local Area Network (LAN). The external interface to Eucalyptus can also be leveraged to be compatible with multiple public clouds, such as Amazon EC2, Sun Cloud…

Finally, there are the number of physical computers in my campus already installed Eucalyptus. Hence, Eucalyptus is the ideal choice for my work.

The block diagram below shows the key components of a sample Private Eucalyptus Cloud.

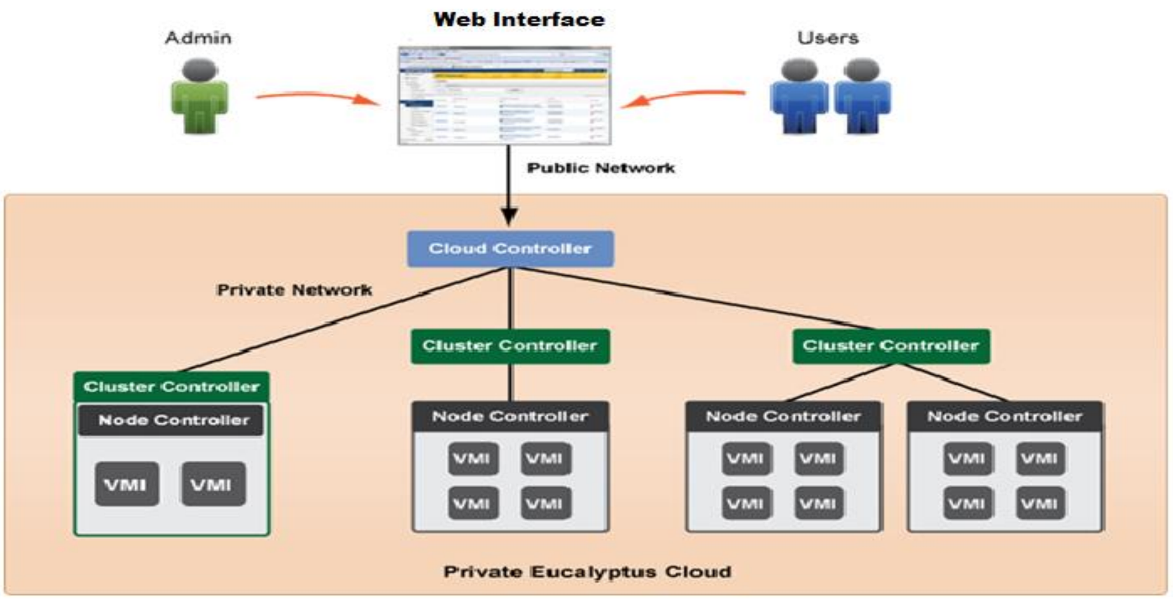


Figure 6 – Private Eucalyptus Cloud Model

The whole system can be controlled via a simple web interface. In Eucalyptus system, there are three main components, namely, Cloud Controller, Cluster Controller and Node Controller. All the components of Eucalyptus will be described in detail in the next section

2.5 Technologies used

- **XML**

XML is designed to transport and store data after choosing deployment. XML acts as an important role in Pegasus operation.

- **Python**

Python is used to create a DAX file in Pegasus.

CHAPTER 3

ARCHITECTURE

3.1. General Architecture

Private Cloud is considered as a suitable model to implement cloud computing in this thesis program due to the following reasons. Private clouds offer various types of users in a variety of business settings: corporate and division offices, business partners, raw-material suppliers, resellers, distributors, and production and supply-chain entities. According to Wikipedia [21], which implies that Private cloud is cloud infrastructure operated solely for a single organization, whether managed internally or by a third-party and hosted internally or externally. They have attracted criticism because users "still have to buy, build, and manage them" and thus do not benefit from less hands-on management, essentially "[lacking] the economic model that makes cloud computing such an intriguing concept".

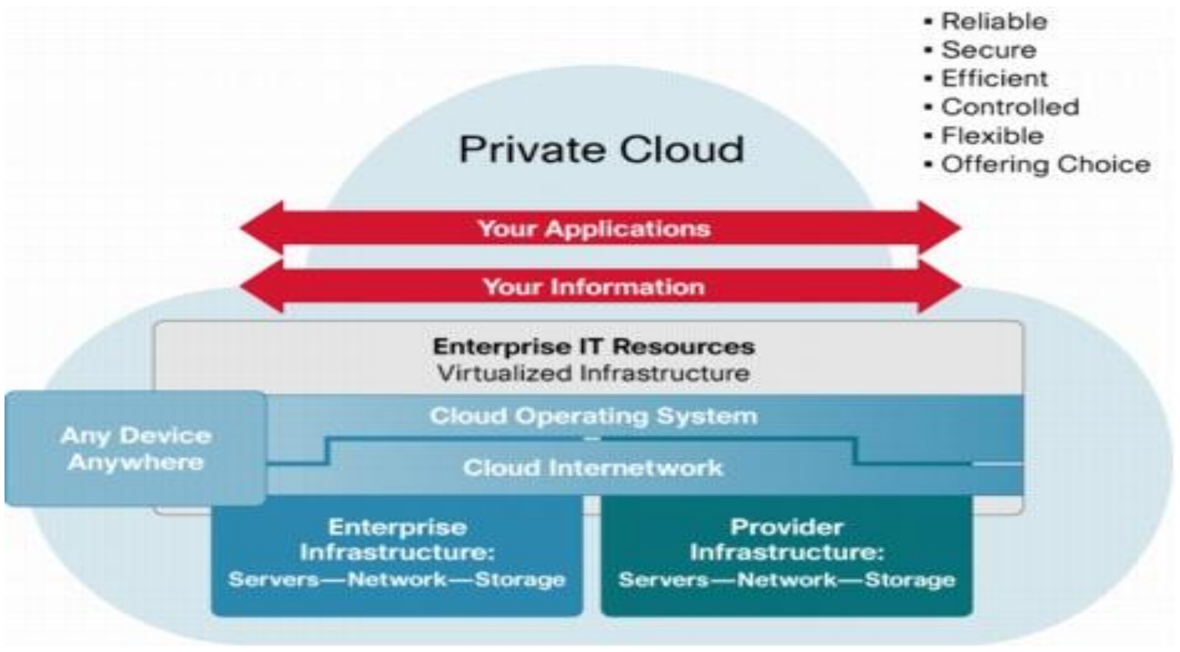


Figure 7 – Private Cloud Model

Furthermore, the private cloud is a new style of computing in which corporate IT infrastructure is available as a ubiquitous, easily accessible, and reliable utility service. Business owners and application owners requesting a new business service can use the infrastructure as a standard service, without requiring staff to undergo any specialized training or the need to understand the complexities of servers, storage, and networks. The private cloud brings the benefits of cloud computing under the control of corporate IT in general and our campus in particular:

- ✓ Available on demand
- ✓ Faster provisioning of business services
- ✓ Reduced cost through economies of scale
- ✓ Flexibility and freedom of choice to run workload and applications in the most efficient and effective places
- ✓ Use-based, pay-as-you-go model
- ✓ Standardized, auditable service levels
- ✓ Capability to work with every application, both current and future, without the need to rewrite applications
- ✓ Secure and controlled by corporate IT

My system of this thesis, based on the general architecture of Eucalyptus cloud setup, yet the system is reduced in a small, sufficient for understanding and imitating easily. All the components of Eucalyptus are still remaining. Those components, then are integrated into my system as the following figure

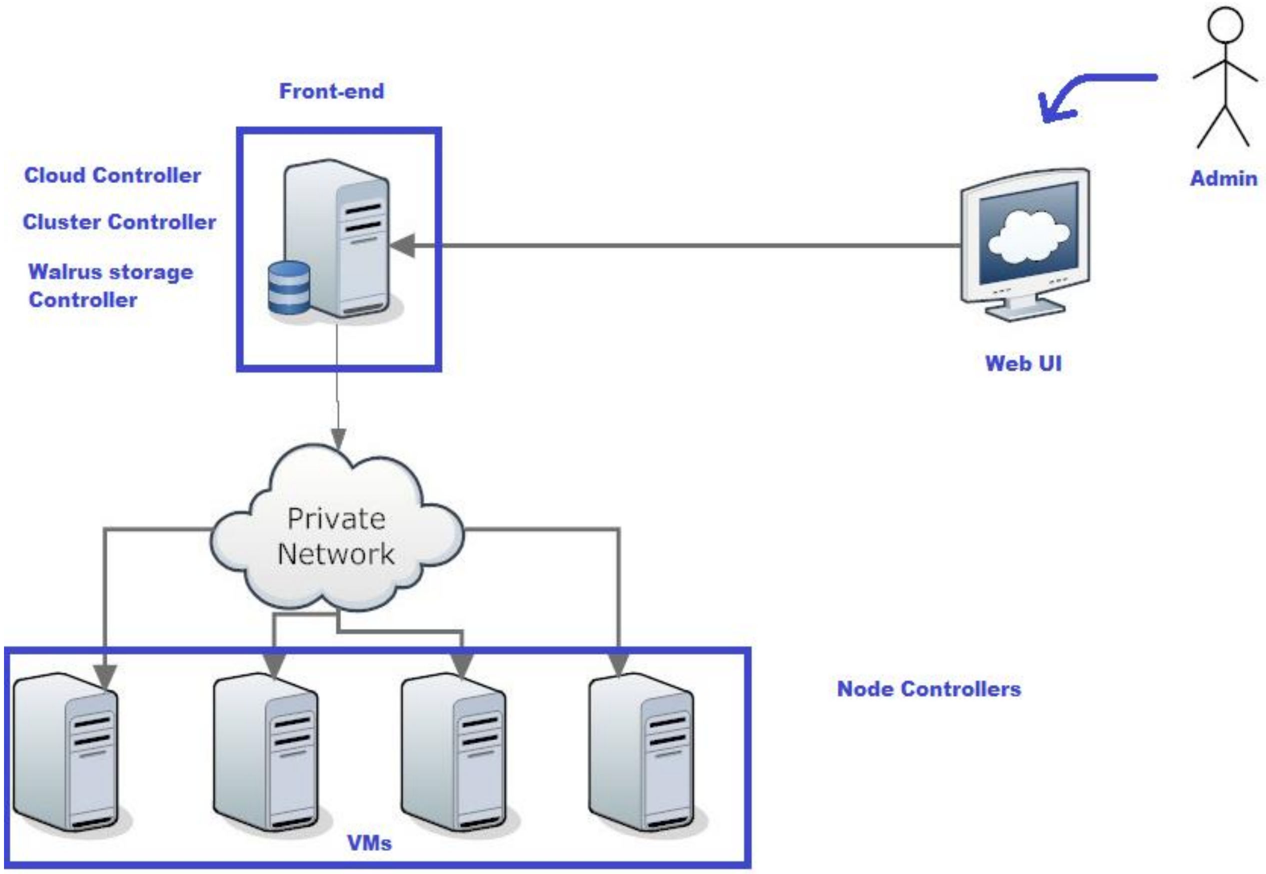


Figure 8 – Implemented Model

In the figure above, the system comprises front-end, node controllers, Web UI. In my system, the front-end act as the vehicle by which the user interacts with the system. The front end is composed of a client computer, or the computer network of an enterprise, and the applications used to access the cloud. In standard Eucalyptus architecture, each physical machine should acts as distinguish roles, which implies there should be three physical machines for Cloud Controllers, Cluster Controllers and Walrus Storage Controllers respectively. Yet in this system, those three main components are installed onto one physical machine. Therefore, the front-end takes responsibilities of all components except Node Controllers.

Node Controllers are installed onto other physical machines. The Node Controllers is the component that generates Virtual Machines depends on available physical resources, it is also responsible for instance start-up, reboot, terminate and clean up.

3.2 Components of a Eucalyptus based Cloud

3.2.1 Node Controller (NC)

Node Controller runs on each node and controls the life cycle of instances running on the node. The NC interacts with the OS and the hypervisor running on the node on one side and the Cluster Controller (CC) on the other side.

NC queries the Operating System running on the node to discover the node’s physical resources – the number of cores, the size of memory, the available disk space and also to learn about the state of VM instances running on the node and propagates this data up to the CC.

Functions:

- Collection of data related to the resource availability and utilization on the node and reporting the data to CC
- Instance life cycle management

3.2.2 Cluster Controller (CC)

CC manages one or more Node Controllers and deploys/manages instances on them. CC also manages the networking for the instances running on the Nodes under certain types of networking modes of Eucalyptus.

CC communicates with Cloud Controller (CLC) on one side and NCs on the other side.

Functions:

- To receive requests from CLC to deploy instances
- To decide which NCs to use for deploying the instances on

3.2.3 Walrus Storage Controller (WS3)

WS3 provides a persistent simple storage service using REST and SOAP APIs compatible with S3 APIs

Functions:

- Storing the machine images
- Storing snapshots
- Storing and serving files using S3 API

WS3 should be considered as a simple file storage system.

3.2.4 Storage Controller (SC)

SC provides persistent block storage for use by the instances. This is similar to the Elastic Block Storage (EBS) service from AWS (Amazon Web Service).

Functions:

- Creation of persistent EBS devices
- Providing the block storage over iSCSI (internet SCSI) [22] **and AoE** (ATA over Ethernet) [23] protocol to the instances.

3.2.5 Cloud Controller (CLC)

The Cloud Controller (CLC) is the front end to the entire cloud infrastructure. CLC provides an EC2/S3 compliant web services interface to the client tools on one side and interacts with the rest of the components of the Eucalyptus infrastructure on the other side. CLC also provides a web interface to users for managing certain aspects of cloud infrastructure.

Functions:

- Monitor the availability of resources on various components of the cloud infrastructure, including hypervisor nodes that are used to actually provision the instances and the cluster controllers that manage the hypervisor nodes

- Resources arbitration – deciding which clusters will be used for provisioning the instances.

- Monitoring the running instances

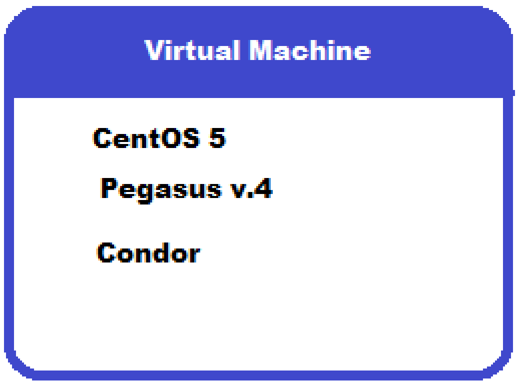


Figure 9 – Inside Virtual Machines

In short, CLC has a comprehensive knowledge of the availability and usage of resources in the cloud and the state of the cloud.

Each Virtual Machine in my system is configured to act as an independent working environment; all VMs are worker node where the workflow will be executed. The CentOS 5 is chosen as main operating system of instances. Besides that, in each instance, workflow applications, such as Pegasus, Condor and prerequisites softwares are also installed.

3.3 Related software to Computing as a Service on Cloud Computing System

The tools implemented in my system are: Pegasus, Condor-DAGMan.

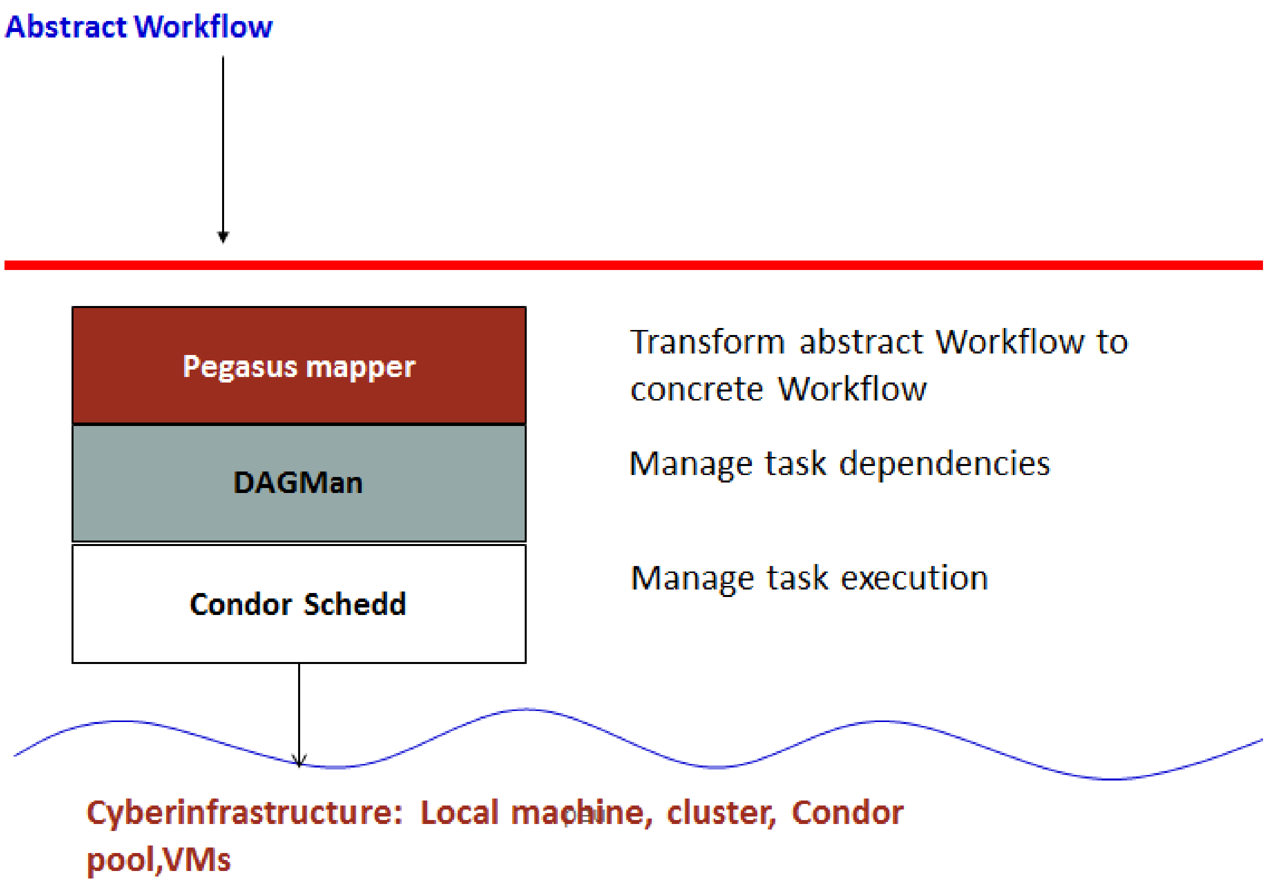

Abstract Workflow
Pegasus mapper
DAGMan
Condor Schedd
Transform abstract Workflow to concrete Workflow
Manage task dependencies
Manage task execution
Cyberinfrastructure: Local machine, cluster, Condor pool,VMs

Figure 10 – Workflow processing steps

The above figure displays Workflow Management System with three main Workflow Applications

- **Mapper (Pegasus Mapper):** Generates an executable workflow based on an abstract workflow provided by the user or workflow composition system. It finds the appropriate software, data, and computational resources required for workflow execution. The Mapper can also restructure the workflow to optimize performance and adds transformations for data management and provenance information generation.

- **Execution Engine (DAGMan):** Executes the tasks defined by the workflow in order of their dependencies. DAGMan relies on the resources (compute, storage and network) defined in the executable workflow to perform the necessary actions.

- **Task manager (Condor Schedd):** manages individual workflow tasks: supervises their execution on local and remote resources.

Abstract workflow is a useful and important part in solving a complex workflow using workflow applications. This enables workflows to be automatically being able to run across variable environments. For complex workflows, abstraction also helps scientists to express their workflows at a higher level without being concerned about the details of how individual jobs are invoked or how data is transferred between jobs. Pegasus Workflow Management System (WMS) [24] is the software used to translate abstract workflow into concrete workflow to be able to be executed. Together with Pegasus, Condor, DAGMan are also selected to deploy in my work.

Technical features of these softwares will be assessed through this section.

Custom virtual machine images containing specific operating system, services, and configuration can be constructed with workflow applications as follow

3.3.1 Pegasus

Pegasus Workflow Management System [WMS] is a configurable system for mapping and executing abstract application workflows over a wide range of execution environment including a laptop, a Grid, a commercial or academic cloud, in this work, it is campus cluster, IUCloud, and the cloud will be discussed in detail in the next chapter. One workflow can run on a single system or across a heterogeneous set of resources, from this point; Pegasus makes use of the power of cloud system. Pegasus can run workflows ranging from just a few computational tasks up to 1 million.

Pegasus WMS bridges the scientific domain and the execution environment by automatically mapping high-level workflow descriptions onto distributed resources. It automatically locates the necessary input data and computational resources for workflow execution. Pegasus enables scientist to construct workflows in abstract terms without worrying about the details of the underlying execution environment or the particulars of the low-level specifications required by the middleware (Condor, Globus). The input to Pegasus is a description of the abstract workflow in XML format.

In summary, the main role of Pegasus is take an abstract workflow (DAX) as an input and generates an executable workflow (DAG), which implies that, DAG [25] is concrete workflow and DAX is DAG in XML language, the generation function could be wrote in Perl, Python or Java programming language.

The black-diamond example, which is similar to CharacterCount Workflow, will be taken in term of explanation the operation of Pegasus

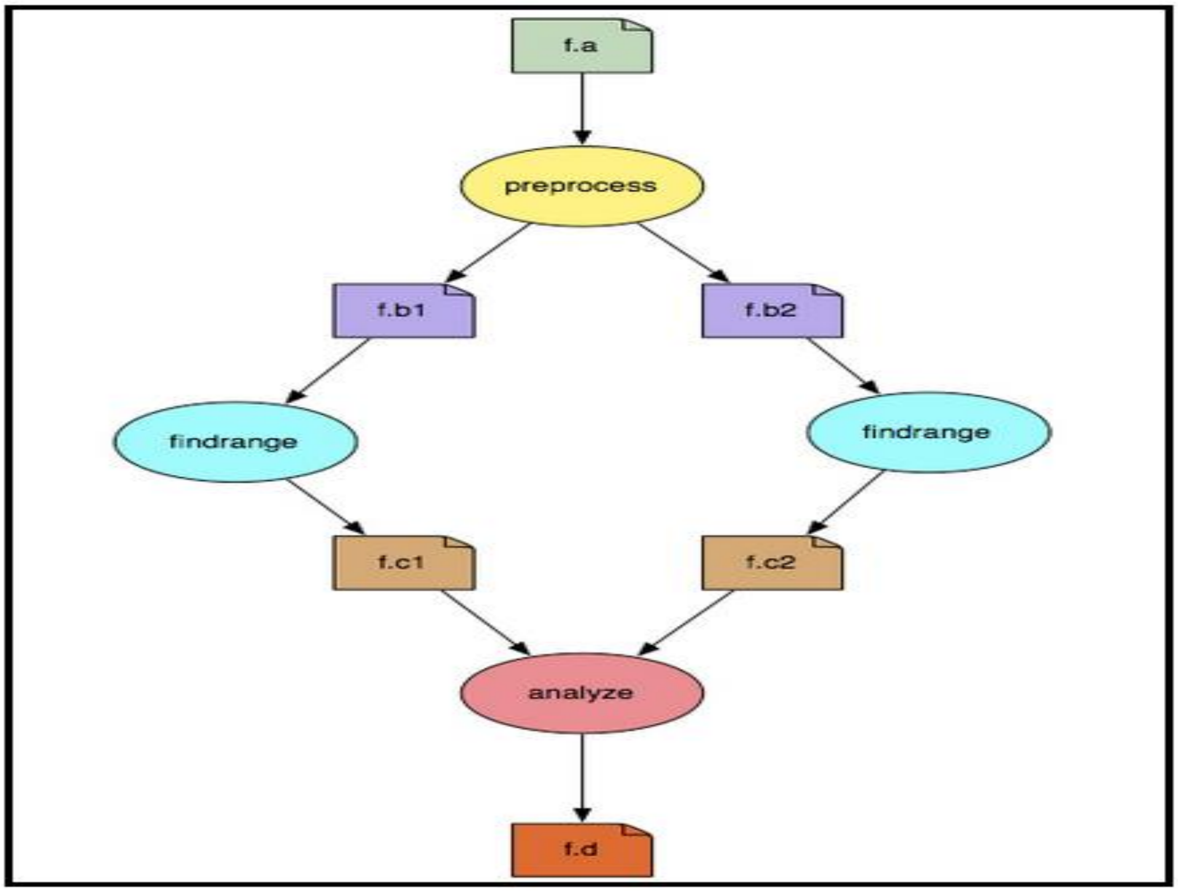


Figure 11 – Abstract Workflow

The figure above shows the workflow in DAX type, which all resource are logically located and all the jobs are placed in logical order.

There are Header and 3 important sections that should be attached in DAX generator.

Header would include xml version, name of dag file will be generated.

- **Section 1**: Files may be used in the workflow, file location
- **Section2** : list all the job will be executed
- **Section 3**: relationship between jobs, parents or child

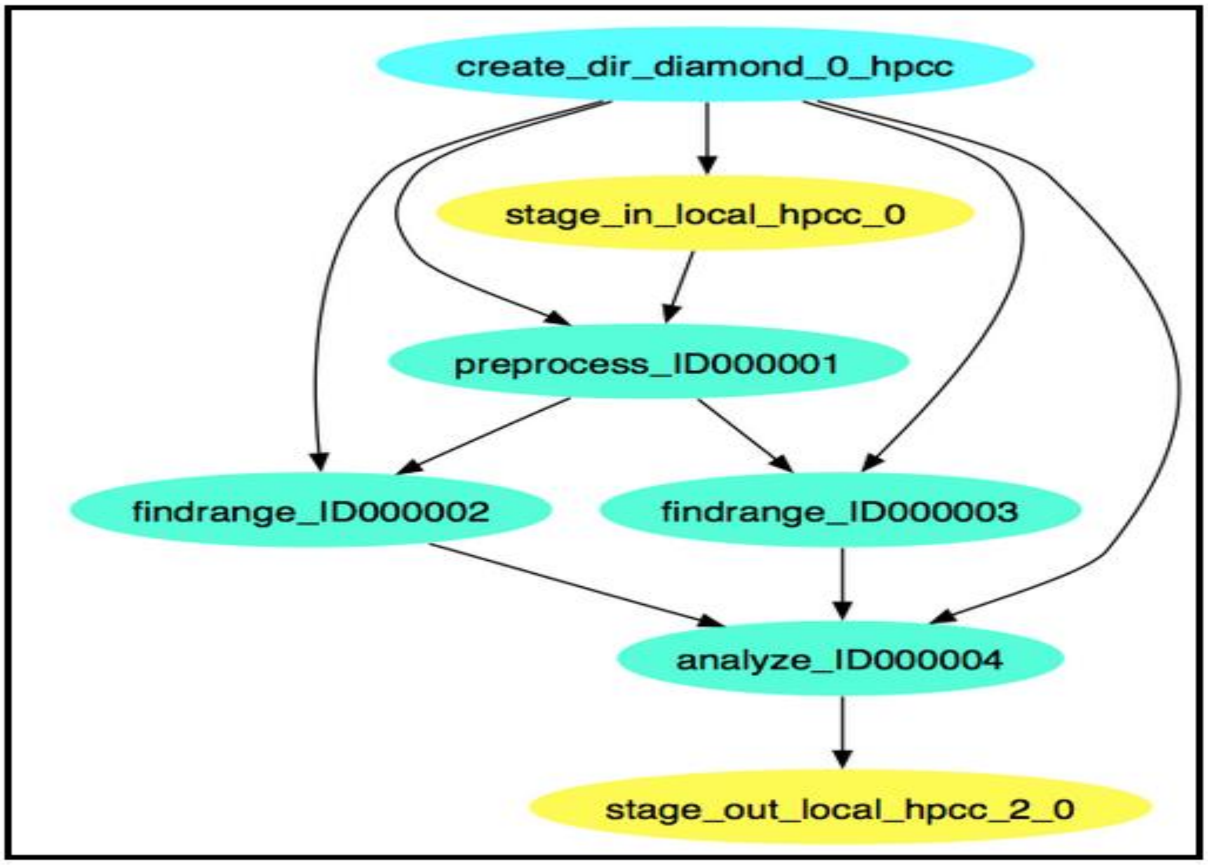
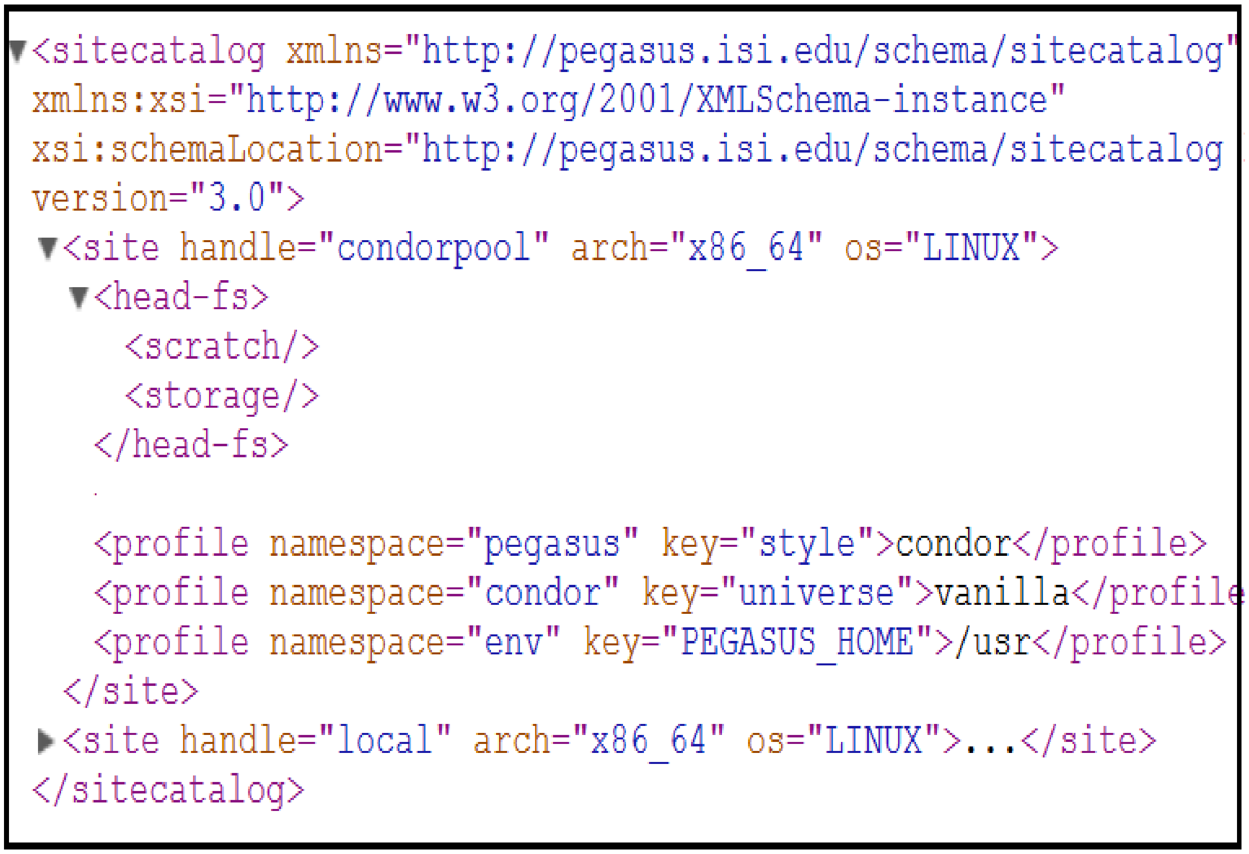


Figure 12– Concrete Workflow

The above graph is the DAG after generated from DAX. In DAG, all the specific resources are located. For example, the physical locations of executable files are specified (Stage_in process), similar with the output location (Stage_out process), and the order of executed jobs also.

The other important element of Pegasus is Site Catalog. The Site Catalog describes the compute resources (which are often clusters) that we intend to run the workflow upon



```
▼<sitecatalog xmlns="http://pegasus.isi.edu/schema/sitecatalog"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://pegasus.isi.edu/schema/sitecatalog
 version="3.0">
  ▼<site handle="condorpool" arch="x86_64" os="LINUX">
    ▼<head-fs>
        <scratch/>
        <storage/>
     </head-fs>

     <profile namespace="pegasus" key="style">condor</profile>
     <profile namespace="condor" key="universe">vanilla</profile
     <profile namespace="env" key="PEGASUS_HOME">/usr</profile>
   </site>
  ▶<site handle="local" arch="x86_64" os="LINUX">...</site>
 </sitecatalog>
```

Figure 13 – Site catalog

Described below are some of the entries in the site catalog.

1. **Site** - A site identifier.

2. **Replica-catalog** - URL for a local replica catalog (LRC) to register your files in. Only used for RLS implementation of the RC. This is optional

3. **File Systems** - Info about file systems mounted on the remote clusters head node or worker nodes. It has several configurations

   - **head-fs/scratch** - This describe the scratch file systems (temporary for execution) available on the head node

   - **head-fs/storage** - This describes the storage file systems (long term) available on the head node

   - **worker-fs/scratch** - This describe the scratch file systems (temporary for execution) available on the worker node

   - **worker-fs/storage** - This describes the storage file systems (long term) available on the worker node

4. **Arch, os, osrelease, osversion, glibc** - The arch/os/osrelease/osversion/glibc of the site. OSRELEASE, OSVERSION and GLIBC are optional

ARCH can have one of the following values X86, X86_64, SPARCV7, SPARCV9, AIX, and PPC. The suitable value for my work is X86

5. **Profiles** - One or many profiles can be attached to a pool.

One example is the environments to be set on a remote pool.

To use this site catalog the follow properties need to be set:

- **pegasus.catalog.site=XML3**

- **pegasus.catalog.site.file=<*path to the site catalog file*>**

Pegasus has a number of features that contribute to its usability and effectiveness.

- **Portability / Reuse**

  User created workflows can easily be run in different environments without alteration. The same workflow can run on a single system or across a heterogeneous set of resources.
- **Performance**

  Pegasus makes use of large scale of computational resources, such as, grids, clouds, thus, the performance of Pegasus is undoubted.
- **Scalability**

  Pegasus can easily scale both the size of the workflow, and the resources that the workflow is distributed over. Pegasus runs workflows ranging from just a few computational tasks up to 1 million. The number of resources involved in executing a workflow can scale as needed without any impediments to performance.
- **Data Management**

  Pegasus handles replica selection, data transfers and output registrations in data catalogs. The Pegasus planner adds these tasks to a workflow as auxiliary jobs.
- **Reliability**

  Jobs and data transfers are automatically retrieved in case of failures. Debugging tools such as Pegasus-analyzer helps the user to debug the workflow in case of non-recoverable failures.
- **Error Recovery**

  When errors occur, Pegasus tries to recover when possible by retrying tasks, by retrying the entire workflow, by providing workflow-level checkpoint, by re-mapping portions of the workflow, by trying alternative data sources for staging data, and, when all else fails, by providing a rescue workflow containing a description of only the work that remains to be done.

Operating Environments

Pegasus workflows can be deployed across a variety of environments:

- **Local execution**
  Workflow can be executed on a single computer with internet access. This is a quicker method for the users do not need to gain access to multiple resources in order to execute a workflow.
- **Condor Pools**
  Condor is a specialized workload management system for compute-intensive jobs, Condor queues workflows, schedules, and monitors the execution of each workflow.
- **Grids**
  Pegasus WMS is entirely compatible with Grid computing. Grid computing relies on the concept of distributed computations.
- **Clouds**
  Cloud computing uses a network as a means to connect a Pegasus end user to distributed resources that are based in the cloud.

However, the environment in my work is condor pool on cloud, which will be described in detail in Chapter 5.

3.3.2 Condor-DAGMan

In my work, the job submitted by end-user will be passed to one physical computer that acts as Submit Host following below Figure-15. Then the input file, at that time the DAG file will be taken by Condor Master, all the worker nodes are VMs (Virtual Machines) will execute the job sent by Condor Master.

Condor

Condor is a software system that creates a High-Throughput Computing (HTC) [26] environment. It effective utilizes the computing power of workstations that communicate over a network. Condor can manage a dedicated cluster of workstations. In the term of solving a workflow, after an abstract workflow has been translated into concrete workflow, the workflow then submitted to Condor. Condor finds an available machine on the cloud and begins running the job there. The power of Condor is that, it can detect whether the machine is idle or not, if yes, it migrate the jobs to there from its checkpoint. Condor continues the job on the new machine from precisely where it left off.

In Condor pool, all machines provides information about their own resources, such as, available RAM memory, CPU type, CPU speed, virtual memory size, physical location and current load average, as a resource offer ad. A user specifies resource requirements followed his/her needs when submitting a job. Condor acts as a broker by matching and ranking resource offer ads with resource request ads, making certain that all requirements in both ads are satisfied. In general speaking, Condor is very effective software.

DAGMan (Directed Acyclic Graph Manager)

There are many approaches about Condor, DAGMan [27] recently in researcher community, thus, this section does not go in deeply about DAGMan, but only its main features.

DAGMan is built to use Condor for job execution, and it can submit jobs to both the local batch system and remote grid or cloud systems with equal ease. Condor finds machines for the execution of programs, but it does not schedule programs (jobs) based on dependencies. The Directed Acyclic Graph Manager (DAGMan) is a meta-scheduler for Condor jobs. DAGMan submits jobs to Condor in an order represented by a DAG and processes the results. An input file defined prior to submission describes the DAG, and a Condor submits description file for each program in the DAG is used by Condor.

DAGMan seeks to run as many jobs as possible in parallel, given the constraints of their parents or child relationships, if there are sufficient resources available… The Inter-Job Dependencies section describes DAGMan in great detail.

In general, the goal of DAGMan is to automate the submission and management of complex workflows involving many jobs, with a focus on reliability and fault-tolerance in the face of a variety of errors. In the simplest case, DAGMan can be used to ensure that two jobs execute sequentially.

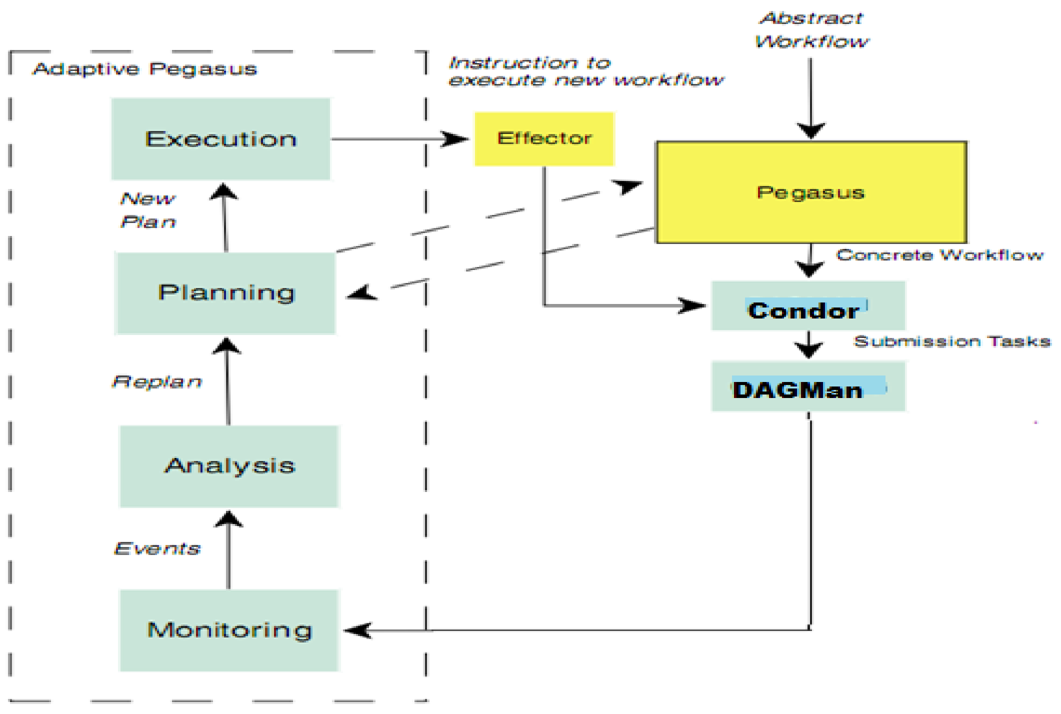


Figure 14 – Pegasus and Condor

The above figure explains the whole procedure in executing an abstract workflow. In summary, there are three steps to solve a problem

Step01: Abstract workflow is translated into concrete workflow by Pegasus

Step02: DAGMan make sure that the concrete workflow is executed in correct way

Step03: Condor detects available resource to solve the workflow.

3.4. Mapping workflows onto cloud resources

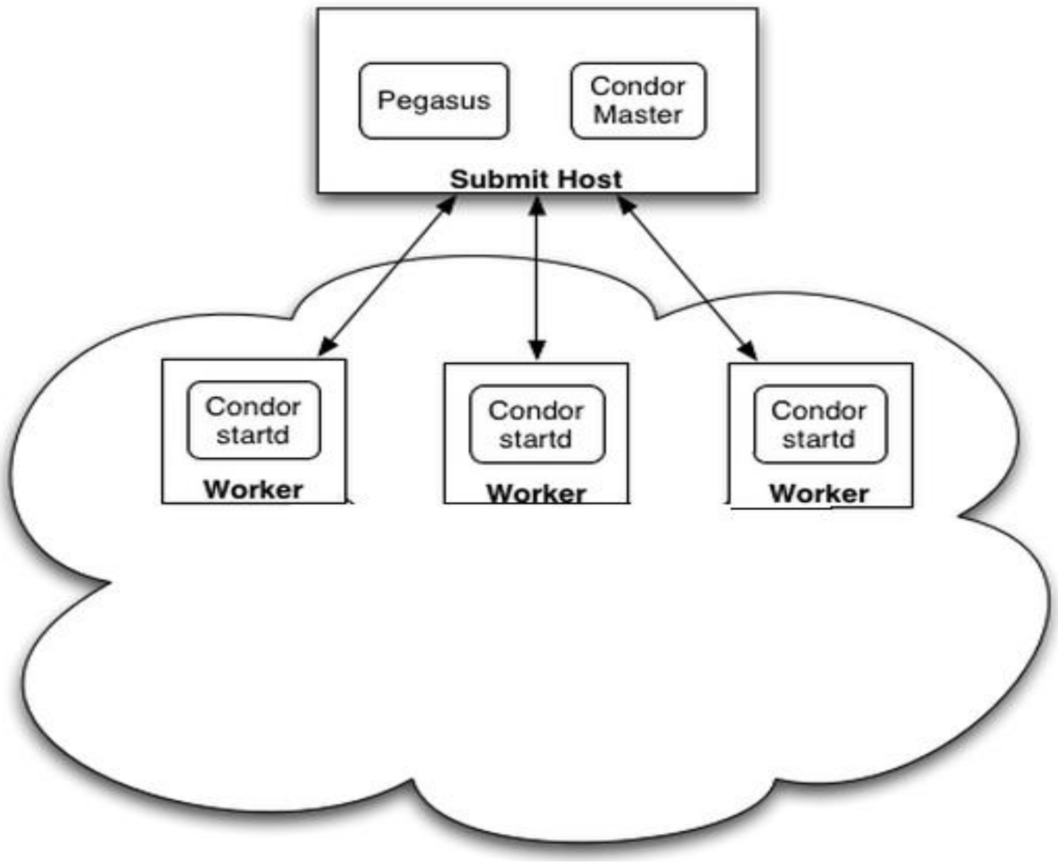


Figure 15 – Workflow Management System Model

On the front-end, main workflow tools described in previous section are also deployed. Front-end will acts as a submit host and VMs are workers following the above figure. The differences between submit host and workers is that the submit host includes all the condor daemon,

- condor_master - controls other daemons
- condor_collector - Collects information from other machines
- condor_negotiator - Performs matchmaking
- condor_schedd - Maintains queue of jobs

In the VMs, only condor_startd daemon is attached to reduce the capacity of instances

- condor_startd - Starts, stops, suspends jobs

CHAPTER 4

IMPLEMENTATION

4.1 Installation Eucalyptus on the private cloud computing

According to the Eucalyptus official website, there are two ways to install Eucalyptus, namely installing by binary source and installing by source package. In this chapter, important steps in installing Eucalyptus from binary source are examined. For detail installation instruction, please go to Appendix section.

Firstly, we have to import the version of Eucalyptus by below command

```
export VERSION=2.0.3
```

At currently, version 2.0.3 is the latest version.

Prerequisites must be installed before main services are deployed following these steps:

- Front-end, node(s), and client machine system clocks are synchronized (e.g., using NTP).

```
Yum install –y ntp

Ntpdate pool.ntp.org
```

- Front end needs java, command to manipulate a bridge, and the binaries for dhcp server (do not configure or run dhcp server on the CC):

```
Yum install –y java-1.6.0-openjdk ant ant-nodeps dhcp bridge-utils perl-Convert-ASN1.noarch scsi-target-utils httpd
```

- Node has a fully installed and configured installation of Xen that allows controlling the hypervisor via HTTP from localhost.

```
Yum install –y xen
```

Then, we have to make sure that the front-end, node(s), and client machines clocks are synchronized. Besides, dhcp server needs to be installed on front-end machines. The following command is to be used to install main services of Eucalyptus

```
Yum install eucalyptus-cloud eucalyptus-cc eucalyptus-walrus eucalyptus-sc
```

And node controllers:

```
Yum install eucalyptus-nc
```

After finishing installation of Eucalyptus on front-end machines, the next job to do is start up the services by following command

```
/etc/init.d/eucalyptus-cloud start
/etc/init.d/eucalyptus-cc start
```

On node controllers

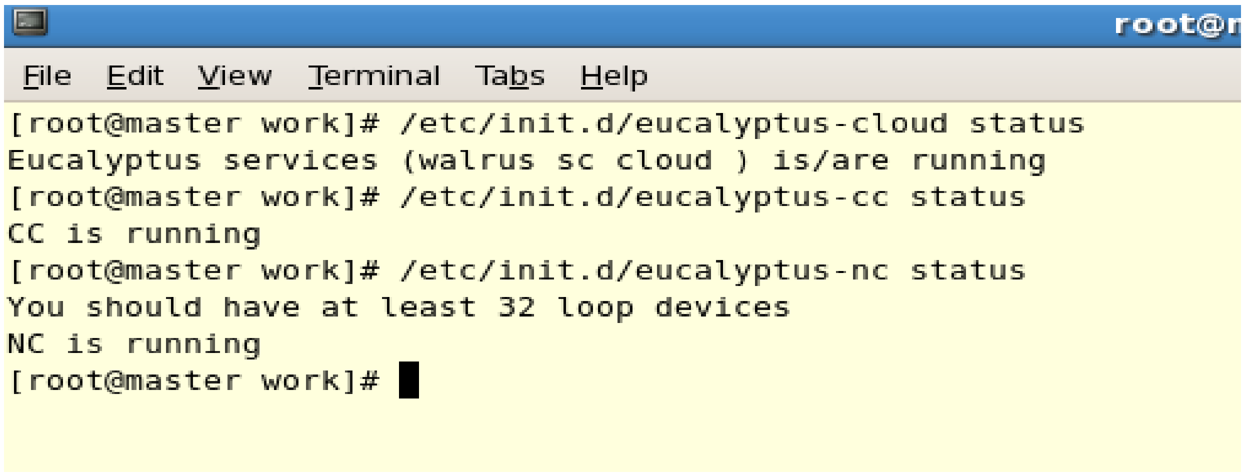
```
/etc/init.d/eucalyptus-nc start
```

Now start up your Eucalyptus services. On the front-end:

```
/etc/init.d/eucalyptus-cloud start

/etc/init.d/eucalyptus-cc start
```

On the node:

```
/etc/init.d/eucalyptus-nc start
```



```
[root@master work]# /etc/init.d/eucalyptus-cloud status
Eucalyptus services (walrus sc cloud ) is/are running
[root@master work]# /etc/init.d/eucalyptus-cc status
CC is running
[root@master work]# /etc/init.d/eucalyptus-nc status
You should have at least 32 loop devices
NC is running
[root@master work]#
```

Figure 16 – Cloud Status

4.2 Deploy workflow tools into Images and Front-end

Pegasus and Condor are deployed on both Front-end and running instances.

Pegasus

Some dependencies are required before setup Pegasus

- Java version >= 1.6
- Python version >= 1.4

Pegasus can be easily install via Pegasus repository file

```
Yum wget –O /etc/yum.repos.d/Pegasus.repo
http://download.pegasus.isi.edu/wms/download/rhel/5/pegasus.repo
```

Search for, and install Pegasus:

```
# yum search pegasus
pegasus.x86_64: Workflow management system for Condor, grids, and clouds
# yum install pegasus
```

Condor

Condor is recommended to install via yum command.

```
cd /etc/yum.repos.d
wget http://research.cs.wisc.edu/condor/yum/repo.d/condor-stable-rhel5.repo
```

```
yum install condor
```

Start Condor daemons

```
service condor start
```

Before uploaded onto the cloud, OS images need to be modified, namely that all the tools such as Pegasus and Condor must be already installed. In this system, CentOS is chosen as a main OS for running instances in the cloud. The image needs to be altered so that instance running on this image can automatically join into the Condor pool and become a worker node in the system, for detail of modifying image instruction, the following commands are used to modify image.

Mounting the root filesystem

Locate the machine image that needs to be altered and mount the image, this contains the root filesystem:

```
$ mkdir <workingdir>
$ mkdir <workingdir>/proc
$ mkdir <workingdir>/dev
# mount -o loop <myimage>.img <workingdir>
# mount -o bind /proc <workingdir>/proc
# mount -o bind /dev <workingdir>/dev
```

With the filesystem and /proc and /dev mounted (this will enable more tools to function correctly), install the required packages when chroot'ed to the mountpoint. All the commands used to install Pegasus and Condor is similar to the command used with front-end above.

Condor Pool

The next step is to configure all started instances automatically join in the condor pool. In this work, the condor pool will be implemented following Figure 15 above. The front-end acts as Central Manager, which controls the condor pool, thus main daemons such as condor_master, condor_negotiator, condor_schedd, will be attached in front-end machine, all worker nodes only need condor_startd to execute jobs. In this program, the domain **master.iu.vn** is taken as central manager domain; worker node domain will be **slaveX.iu.vn** correspondingly. For daemons configuration, the file **condor_config.local** will be altered.

With Central Manger, two points need to be changed.

```
## What machine is your central manager?

CONDOR_HOST = $(FULL_HOSTNAME)

##  The list is a comma or space separated list of subsystem names

DAEMON_LIST = COLLECTOR, MASTER, NEGOTIATOR, SCHEDD
```

The hostname of central manager can be changed via following command

```
hostname master.iu.vn
```

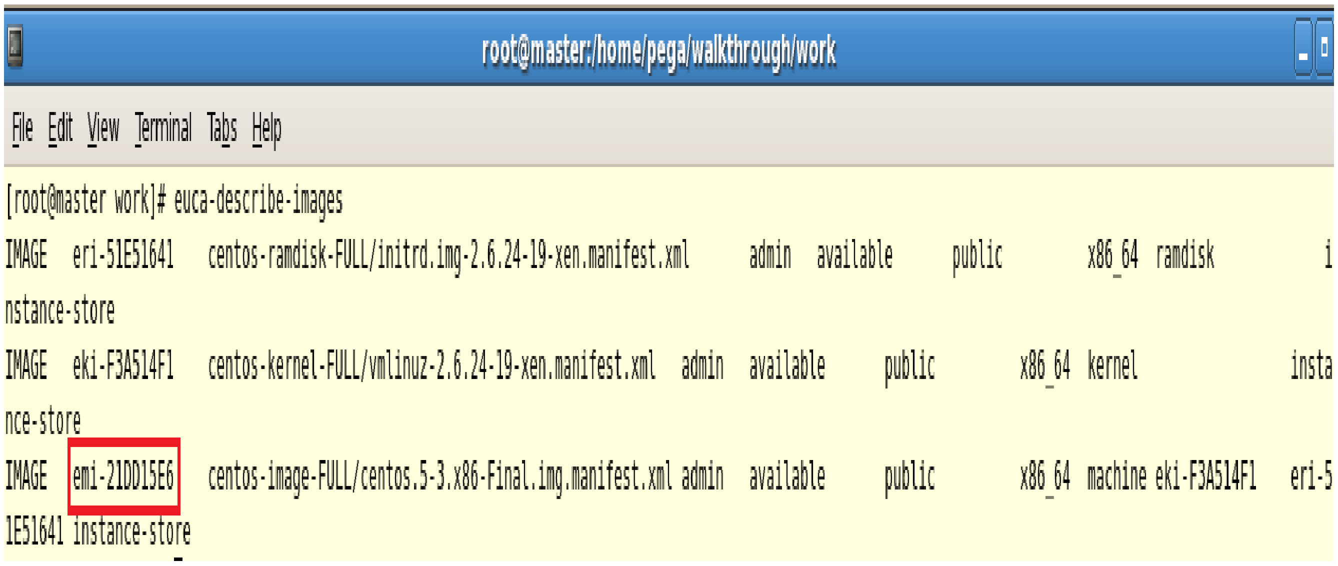
With instance – worker node, condor host must be pointed to Central Manager with hostname master.iu.vn, only daemons condor_startd is need

```
## What machine is your central manager?

CONDOR_HOST = master.iu.vn

##  The list is a comma or space separated list of subsystem names

DAEMON_LIST = STARTD
```

After all the softwares are installed, image will be uploaded to the cloud to work as independent environment. Uploading command should be referenced via Appendix.

All the information of uploaded images can be displayed with a simple command

```
Euca-describe-images
```


root@master:/home/pega/walkthrough/work
File Edit View Terminal Tabs Help

```
[root@master work]# euca-describe-images
IMAGE   eri-51E51641   centos-ramdisk-FULL/initrd.img-2.6.24-19-xen.manifest.xml     admin   available     public       x86_64  ramdisk          i
nstance-store
IMAGE   eki-F3A514F1   centos-kernel-FULL/vmlinuz-2.6.24-19-xen.manifest.xml   admin  available     public       x86_64  kernel               insta
nce-store
IMAGE   emi-21DD15E6   centos-image-FULL/centos.5-3.x86-Final.img.manifest.xml admin  available     public       x86_64  machine eki-F3A514F1  eri-5
1E51641 instance-store
```

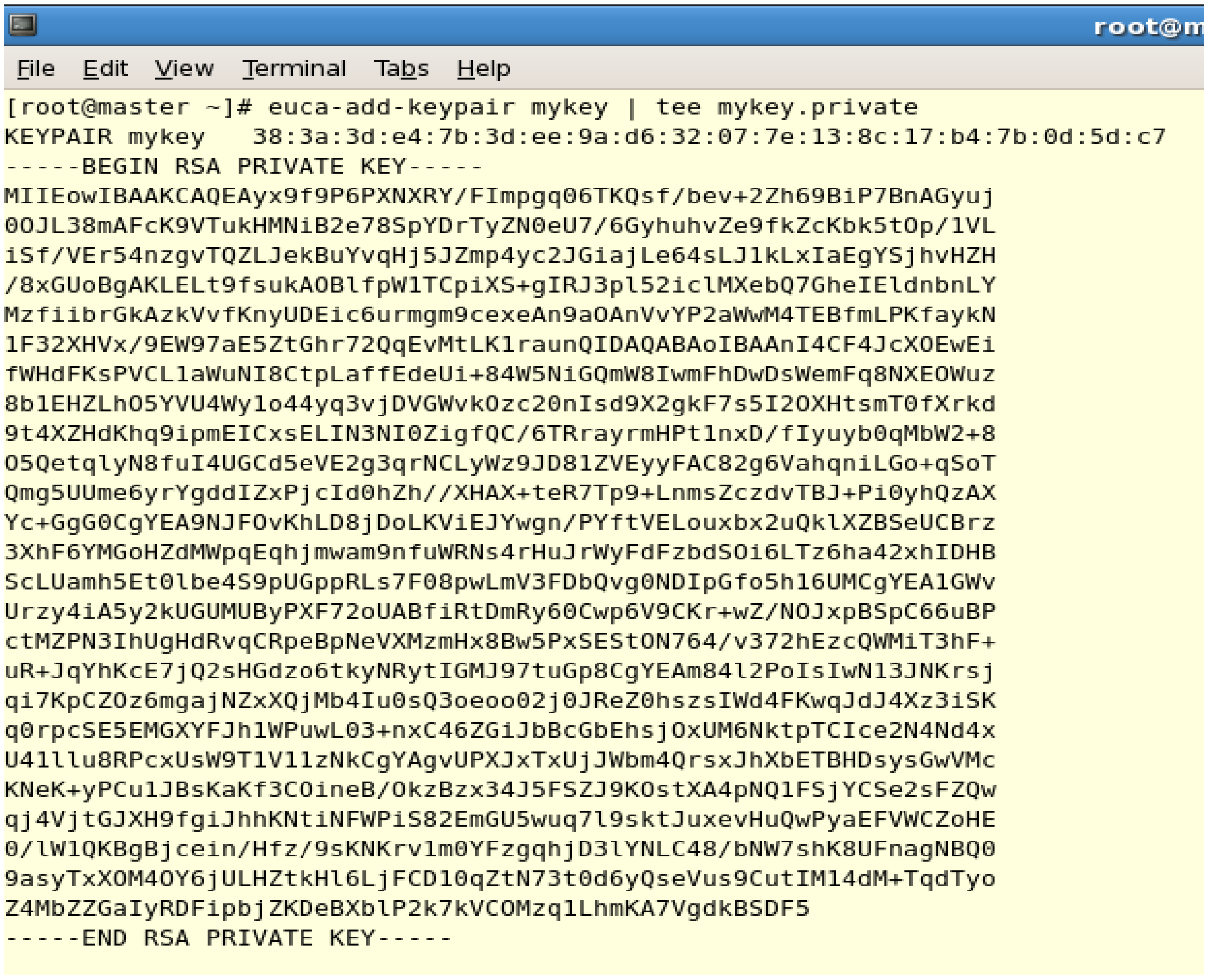
Figure 17 – Running Instance

Before starting up instance with specific Image id, for example, emi-21DD15E6, the keypairs for instance must be created first.

Keypairs can be created with the following command

```
Euca-add-keypair mykey | tee mykey.private
```

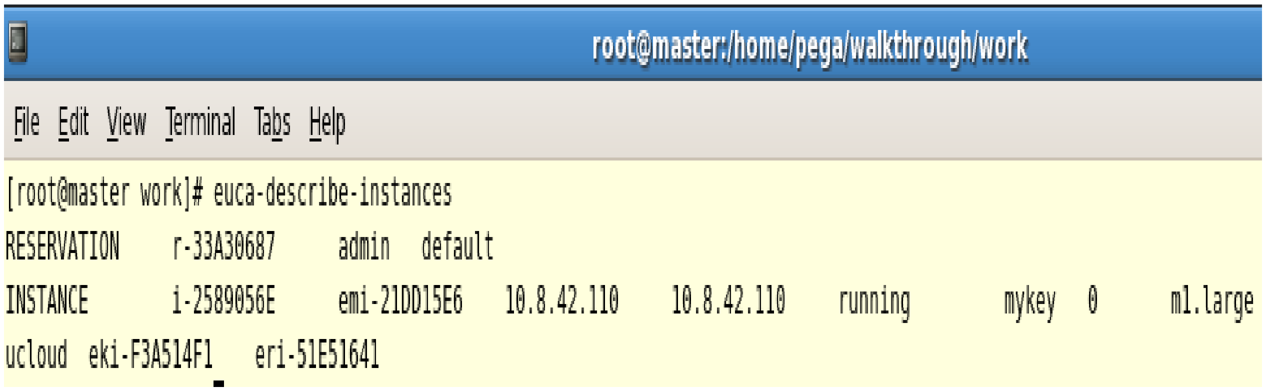
A pair of keys is created; one public key, stored in Eucalyptus, and one private key stored in the file mykey.private and printed to standard output.



```
[root@master ~]# euca-add-keypair mykey | tee mykey.private
KEYPAIR mykey   38:3a:3d:e4:7b:3d:ee:9a:d6:32:07:7e:13:8c:17:b4:7b:0d:5d:c7
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAyx9f9P6PXNXRY/FImpgq06TKQsf/bev+2Zh69BiP7BnAGyuj
0OJL38mAFcK9VTukHMNiB2e78SpYDrTyZN0eU7/6GyhuhvZe9fkZcKbk5tOp/1VL
iSf/VEr54nzgvTQZLJekBuYvqHj5JZmp4yc2JGiajLe64sLJ1kLxIaEgYSjhvHZH
/8xGUoBgAKLELt9fsukAOBlfpW1TCpiXS+gIRJ3pl52iclMXebQ7GheIEldnbnLY
MzfiibrGkAzkVvfKnyUDEic6urmgm9cexeAn9aOAnVvYP2aWwM4TEBfmLPKfaykN
1F32XHVx/9EW97aE5ZtGhr72QqEvMtLK1raunQIDAQABAoIBAAnI4CF4JcXOEwEi
fWHdFKsPVCL1aWuNI8CtpLaffEdeUi+84W5NiGQmW8IwmFhDwDsWemFq8NXEOWuz
8b1EHZLhO5YVU4Wy1o44yq3vjDVGWvkOzc20nIsd9X2gkF7s5I2OXHtsmT0fXrkd
9t4XZHdKhq9ipmEICxsELIN3NI0ZigfQC/6TRrayrmHPt1nxD/fIyuyb0qMbW2+8
O5QetqlyN8fuI4UGCd5eVE2g3qrNCLyWz9JD81ZVEyyFAC82g6VahqniLGo+qSoT
Qmg5UUme6yrYgddIZxPjcId0hZh//XHAX+teR7Tp9+LnmsZczdvTBJ+Pi0yhQzAX
Yc+GgG0CgYEA9NJFOvKhLD8jDoLKViEJYwgn/PYftVELouxbx2uQklXZBSeUCBrz
3XhF6YMGoHZdMWpqEqhjmwam9nfuWRNs4rHuJrWyFdFzbdSOi6LTz6ha42xhIDHB
ScLUamh5Et0lbe4S9pUGppRLs7F08pwLmV3FDbQvg0NDIpGfo5h16UMCgYEA1GWv
Urzy4iA5y2kUGUMUByPXF72oUABfiRtDmRy60Cwp6V9CKr+wZ/NOJxpBSpC66uBP
ctMZPN3IhUgHdRvqCRpeBpNeVXMzmHx8Bw5PxSEStON764/v372hEzcQWMiT3hF+
uR+JqYhKcE7jQ2sHGdzo6tkyNRytIGMJ97tuGp8CgYEAm84l2PoIsIwN13JNKrsj
qi7KpCZOz6mgajNZxXQjMb4Iu0sQ3oeoo02j0JReZ0hszsIWd4FKwqJdJ4Xz3iSK
q0rpcSE5EMGXYFJh1WPuwL03+nxC46ZGiJbBcGbEhsjOxUM6NktpTCIce2N4Nd4x
U41llu8RPcxUsW9T1V11zNkCgYAgvUPXJxTxUjJWbm4QrsxJhXbETBHDsysGwVMc
KNeK+yPCu1JBsKaKf3COineB/OkzBzx34J5FSZJ9KOstXA4pNQ1FSjYCSe2sFZQw
qj4VjtGJXH9fgiJhhKNtiNFWPiS82EmGU5wuq7l9sktJuxevHuQwPyaEFVWCZoHE
0/lW1QKBgBjcein/Hfz/9sKNKrv1m0YFzgqhjD3lYNLC48/bNW7shK8UFnagNBQ0
9asyTxXOM4OY6jULHZtkHl6LjFCD10qZtN73t0d6yQseVus9CutIM14dM+TqdTyo
Z4MbZZGaIyRDFipbjZKDeBXblP2k7kVCOMzq1LhmKA7VgdkBSDF5
-----END RSA PRIVATE KEY-----
```

Figure 18 – Keypair

After the instance has been started up, it can be login via ssh command with the specific IP address assigned by DHCP server. According to Eucalyptus instruction, there are 2 methods to assign IP to instance, Manual IP or dynamic IP via DHCP server. In this thesis program, I prefer to dynamic IP, which is configured as default.


root@master:/home/pega/walkthrough/work
File Edit View Terminal Tabs Help

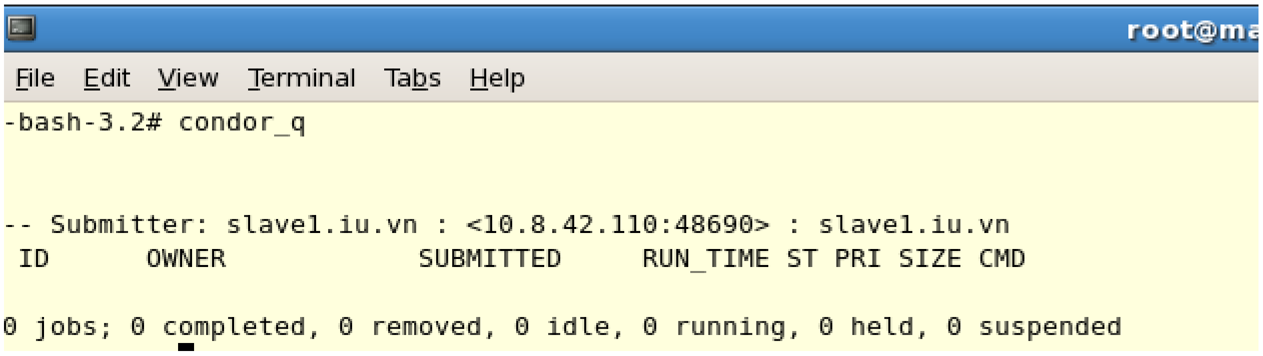
```
[root@master work]# euca-describe-instances
RESERVATION     r-33A30687      admin   default
INSTANCE        i-2589056E      emi-21DD15E6    10.8.42.110     10.8.42.110     running         mykey   0       m1.large
ucloud  eki-F3A514F1    eri-51E51641
```

Figure 19 – Instance Information

We can login to instance with ssh command and private key generated in previous step

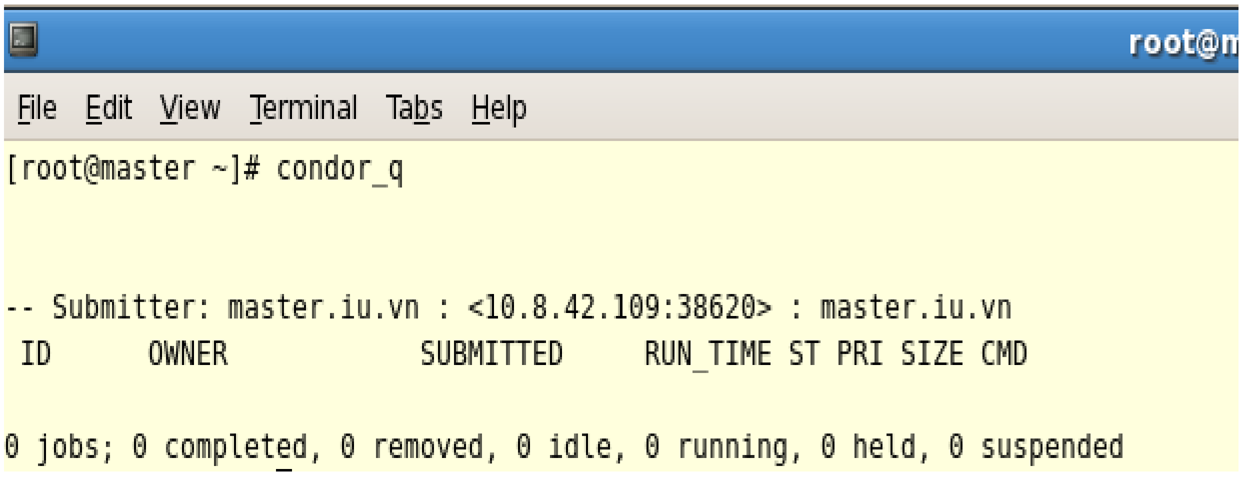
Ssh –I mykey.private root@instance.IP.address

After logging in the instance, it is clear to see that all the deployed tools work properly. The condor tool in VMs automatically joins in condor pool.



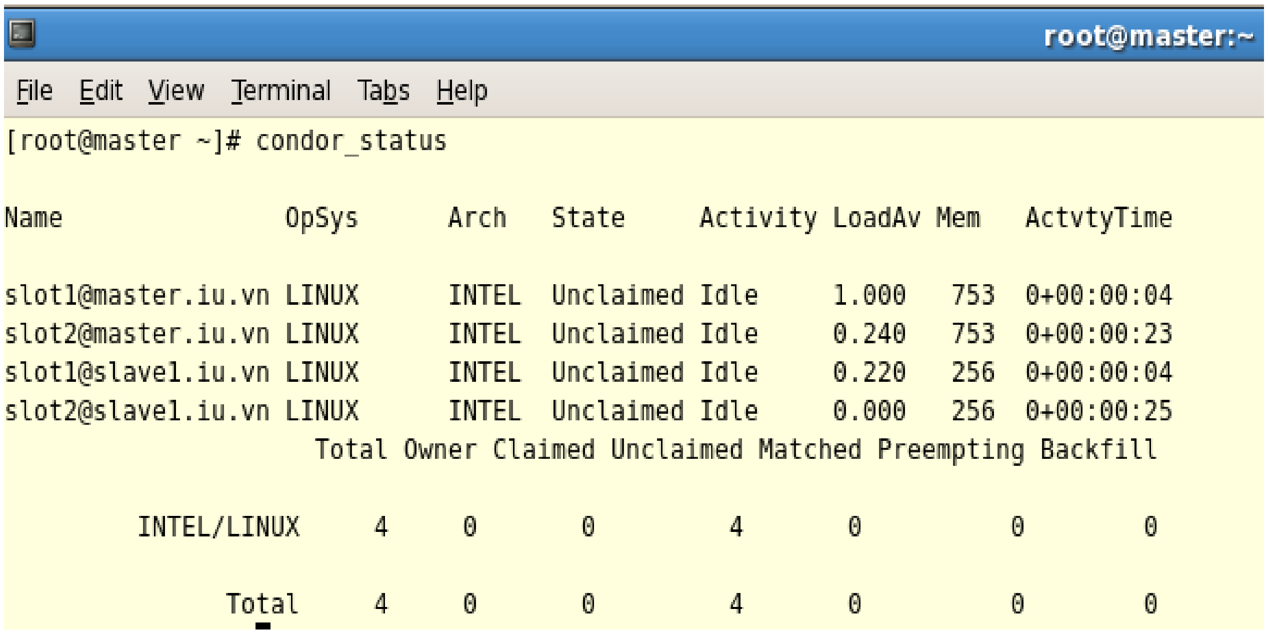
```
-bash-3.2# condor_q


-- Submitter: slave1.iu.vn : <10.8.42.110:48690> : slave1.iu.vn
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD

0 jobs; 0 completed, 0 removed, 0 idle, 0 running, 0 held, 0 suspended
```

Figure 20 – Worker condor



```
[root@master ~]# condor_q


-- Submitter: master.iu.vn : <10.8.42.109:38620> : master.iu.vn
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD

0 jobs; 0 completed, 0 removed, 0 idle, 0 running, 0 held, 0 suspended
```

Figure 21 – Master condor

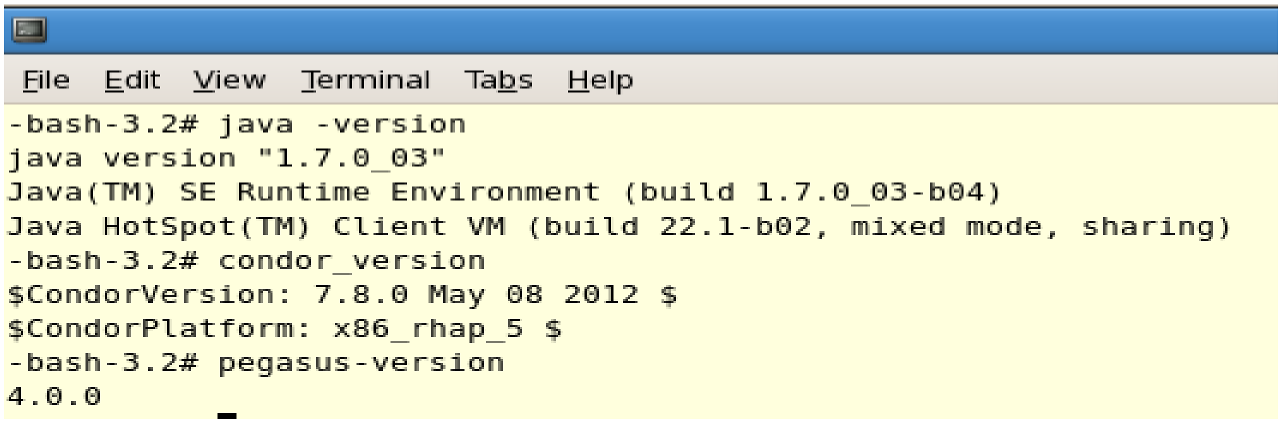


```
[root@master ~]# condor_status

Name               OpSys      Arch   State     Activity LoadAv Mem   ActvtyTime

slot1@master.iu.vn LINUX      INTEL  Unclaimed Idle     1.000   753  0+00:00:04
slot2@master.iu.vn LINUX      INTEL  Unclaimed Idle     0.240   753  0+00:00:23
slot1@slave1.iu.vn LINUX      INTEL  Unclaimed Idle     0.220   256  0+00:00:04
slot2@slave1.iu.vn LINUX      INTEL  Unclaimed Idle     0.000   256  0+00:00:25
                     Total Owner Claimed Unclaimed Matched Preempting Backfill

         INTEL/LINUX     4     0       0         4       0          0        0

               Total     4     0       0         4       0          0        0
```

Figure 22 – Condor pool

The entire worker node are free from resources, the activity will be stated “idle”. Otherwise, the state is “claimed” in case that the resources are being used and the activity is “busy”.

Due to the lack of hardware resources, in this system, I only use one virtual machine in the cloud as a worker node. In the condor pool, there are two worker node, master node and slave1 node, in each node; there are two processing cores represents for two slots, slot1 and slot2.



```
-bash-3.2# java -version
java version "1.7.0_03"
Java(TM) SE Runtime Environment (build 1.7.0_03-b04)
Java HotSpot(TM) Client VM (build 22.1-b02, mixed mode, sharing)
-bash-3.2# condor_version
$CondorVersion: 7.8.0 May 08 2012 $
$CondorPlatform: x86_rhap_5 $
-bash-3.2# pegasus-version
4.0.0
```

Figure 23 – Prerequisite softwares

4.3 Workflow processing

4.3.1. Create the workflow (DAX)

The important things required in solving workflow are abstract workflow, which is described in detail in previous chapter, the others are Site catalog and Pegasus Properties also need to be altered.

For the purpose of demonstration, the CharacterCount problem is chosen; all the jobs are executed following Figure-1 CharacterCount Worflow.

Pegasus supports three programming languages Java, Python and Perl to generate abstract workflow (DAX), in this paper, python is chosen as a main DAX generator. The Python code as below has been edited based on standard format, which is retrieved from Pegasus official website.

```
#!/usr/bin/env python

import sys
import os

# The location of Pegasus is needed for two things:
# 1. to find the Python DAX module
# 2. to find the keg sample binary
if len(sys.argv) != 2:
	print "Usage: %s PEGASUS_HOME" % (sys.argv[0])
	sys.exit(1)
pegasus_home = sys.argv[1]
os.sys.path.insert(0, os.path.join(pegasus_home, "lib/pegasus/python"))

# Pegaus imports
from Pegasus.DAX3 import *

# Create a abstract dag
diamond = ADAG("diamond")

# Add input file to the DAX-level replica catalog
a = File("text.txt")
a.addPFN(PFN("file:///home/pega/demo-test/text.txt", "condorpool"))
diamond.addFile(a)

# Add executables to the DAX-level replica catalog
e_preprocess = Executable(namespace="diamond", name="preprocess", version="4.0",
os="linux", arch="x86_64", installed=False)
e_preprocess.addPFN(PFN("file://" + pegasus_home + "/bin/job/job1.py", "condorpool"))
diamond.addExecutable(e_preprocess)
```

```
e_findrange2 = Executable(namespace="diamond", name="findrange2", version="4.0", os="linux", arch="x86_64", installed=False)
e_findrange2.addPFN(PFN("file://" + pegasus_home + "/bin/job/job2.py", "condorpool"))
diamond.addExecutable(e_findrange2)
e_findrange3 = Executable(namespace="diamond", name="findrange3", version="4.0", os="linux", arch="x86_64", installed=False)
e_findrange3.addPFN(PFN("file://" + pegasus_home + "/bin/job/job3.py", "condorpool"))
diamond.addExecutable(e_findrange3)
e_analyze = Executable(namespace="diamond", name="analyze", version="4.0", os="linux", arch="x86_64", installed=False)
e_analyze.addPFN(PFN("file://" + pegasus_home + "/bin/job/job4.py", "condorpool"))
diamond.addExecutable(e_analyze)

# Add a preprocess job
preprocess = Job(namespace="diamond", name="preprocess", version="4.0")
b1 = File("text2.txt")
b2 = File("text3.txt")
preprocess.addArguments("-a preprocess","-T60","-i",a,"-o",b1,b2)
preprocess.uses(a, link=Link.INPUT)
preprocess.uses(b1, link=Link.OUTPUT)
preprocess.uses(b2, link=Link.OUTPUT)
diamond.addJob(preprocess)

# Add left Findrange job
frl = Job(namespace="diamond", name="findrange2", version="4.0")
c1 = File("FinalText1.txt")
frl.addArguments("-a findrange2","-T60","-i",b1,"-o",c1)

frl.uses(b1, link=Link.INPUT)
frl.uses(c1, link=Link.OUTPUT)
```

```
diamond.addJob(frl)

# Add right Findrange job
frr = Job(namespace="diamond", name="findrange3", version="4.0")
c2 = File("FinalText2.txt")
frr.addArguments("-a findrange3","-T60","-i",b2,"-o",c2)
frr.uses(b2, link=Link.INPUT)
frr.uses(c2, link=Link.OUTPUT)
diamond.addJob(frr)

# Add Analyze job
analyze = Job(namespace="diamond", name="analyze", version="4.0")
d = File("FinalText.txt")
analyze.addArguments("-a analyze","-T60","-i",c1,c2,"-o",d)
analyze.uses(c1, link=Link.INPUT)
analyze.uses(c2, link=Link.INPUT)
analyze.uses(d, link=Link.OUTPUT, register=True)
diamond.addJob(analyze)

# Add control-flow dependencies
diamond.addDependency(Dependency(parent=preprocess, child=frl))
diamond.addDependency(Dependency(parent=preprocess, child=frr))
diamond.addDependency(Dependency(parent=frl, child=analyze))
diamond.addDependency(Dependency(parent=frr, child=analyze))

# Write the DAX to stdout
diamond.writeXML(sys.stdout)
```

The code above has 6 logical sections:

1. **Imports and Pegasus location setup**. The specific location of Pegasus must be declared, hence, DAX3 python module can be imported.
2. **A new abstract dag (DAX) is created.** This is the main DAX object that we will add data, jobs and flow information to. In this example, the name of DAX file is **diamond.xml**
3. **A replica catalog is set up.** The Replica Catalog keeps mappings of logical file ids/names (LFN's) to physical file ids/names (PFN's). In this work, physical file name of the input, text.txt is specified.
4. **Executables are added.** There are four executable files are added, namely Job1, Job2, Job3, Job4. For more detail, Job1 will be called **Preprocess** job, Job2 and Job3 are **Findrange2** and **Findrange3** correspondingly. Job4 is **Analyze** job.
5. **Jobs are added.** The 4 jobs in the Black Diamond in the picture above are added. Arguments are defined, and **uses** clauses are added to list input and output files.
6. **Control flows are set up.** This is the edges in the picture, and defines parent/child relationships between the jobs. When the workflow is executing, this is the order the jobs will run in.

The procedure of solving this problem was described in detail in Chapter 1. In summary, there are four main jobs need to be considered, namely, Job1 will do split job, Job2 and Job3 will concurrently count the two new text files. Job4 will count the total character by do the sum of the output of the two previous jobs. The content of those executable files are as below

Job1

```
#! /usr/bin/env python
#!/usr/bin/python
# open file
f = open ("text.txt","r")

# Read whole file into data
data = f.read()
```

```
#Split text
job2, job3 = data[:len(data)/2], data[len(data)/2:]

# Write new data
f = open ("text2.txt","w")
f.write(job2)
f.close()

f = open ("text3.txt","w")
f.write(job3)
f.close()
#close data
f.close()
```

Job2

```
#! /usr/bin/env python
#!/usr/bin/python
# open file
f = open ("text2.txt","r")

# Read whole file into data
data = f.read()

#Count string
chars_no_spaces = sum([not x.isspace() for x in data])

str2 = "No of characters: " + str(chars_no_spaces)

# final
```

```
f = open ("FinalText1.txt","w")
f.write(str2 + '\n')
f.close()
```

Job3

```
#! /usr/bin/env python
#!/usr/bin/python
# open file
f = open ("text3.txt","r")

# Read whole file into data
data = f.read()

#Count string
chars_no_spaces = sum([not x.isspace() for x in data])
str2 = "No of characters: " + str(chars_no_spaces)

# final
f = open ("FinalText2.txt","w")
f.write(str2 + '\n')
f.close()
```

Job4

```
#! /usr/bin/env python
#!/usr/bin/python
import os
# open file
f1 = open ("FinalText1.txt","r")
```

```
# Read whole file into data
data1 = f1.read()
num1=[int(s) for s in data1.split() if s.isdigit()]
tmp = num1[0]
num1=tmp

# open file
f2 = open ("FinalText2.txt","r")

# Read whole file into data
data2 = f2.read()
num2=[int(s) for s in data2.split() if s.isdigit()]
tmp2=num2[0]
num2=tmp2
total = num1 + num2
str1 = "Number of character is ", total
tmp = str(str1)
str1=tmp
f1.close()
f2.close()

# Write new data
f = open ("FinalText.txt","w")
f.write(str1)
f.close()
```

The abstract workflow in XML format could be generate by the command

```
$ ./create_diamond_dax.py /usr > diamond.xml
```

The command above using python DAX generator to create a DAX file named diamond, the Pegasus bin path should be pointed out in the command. The output file diamond.xml is as below

```
<? xml version="1.0" encoding="UTF-8"?>
<!-- generated: 2012-06-28 12:43:49.555145 -->
<!-- generated by: pega -->
<!-- generator: python -->
<adag xmlns="http://pegasus.isi.edu/schema/DAX"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://pegasus.isi.edu/schema/DAX http://pegasus.isi.edu/schema/dax-
3.3.xsd" version="3.3" name="diamond">
	<file name="text.txt">
		<pfn url="file:///home/pega/demo-test/text.txt" site="condorpool"/>
	</file>
	<executable name="findrange3" namespace="diamond" version="4.0"
arch="x86_64" os="linux" installed="false">
		<pfn url="file:///usr/bin/job/job3.py" site="condorpool"/>
	</executable>
	<executable name="preprocess" namespace="diamond" version="4.0"
arch="x86_64" os="linux" installed="false">
		<pfn url="file:///usr/bin/job/job1.py" site="condorpool"/>
	</executable>
	<executable name="analyze" namespace="diamond" version="4.0" arch="x86_64"
os="linux" installed="false">
		<pfn url="file:///usr/bin/job/job4.py" site="condorpool"/>
	</executable>
	<executable name="findrange2" namespace="diamond" version="4.0"
arch="x86_64" os="linux" installed="false">
		<pfn url="file:///usr/bin/job/job2.py" site="condorpool"/>
	</executable>
	<job id="ID0000001" namespace="diamond" name="preprocess" version="4.0">
		<argument>-a preprocess -T60 -i <file name="text.txt"/> -o <file
name="text2.txt"/> <file name="text3.txt"/></argument>
		<uses name="text.txt" link="input"/>
```

```
                <uses name="text2.txt" link="output"/>
                <uses name="text3.txt" link="output"/>
        </job>
        <job id="ID0000002" namespace="diamond" name="findrange2" version="4.0">
                <argument>-a findrange2 -T60 -i <file name="text2.txt"/> -o <file
name="FinalText1.txt"/></argument>
                <uses name="text2.txt" link="input"/>
                <uses name="FinalText1.txt" link="output"/>
        </job>
        <job id="ID0000003" namespace="diamond" name="findrange3" version="4.0">
                <argument>-a findrange3 -T60 -i <file name="text3.txt"/> -o <file
name="FinalText2.txt"/></argument>
                <uses name="text3.txt" link="input"/>
                <uses name="FinalText2.txt" link="output"/>
        </job>
        <job id="ID0000004" namespace="diamond" name="analyze" version="4.0">
                <argument>-a analyze -T60 -i <file name="FinalText1.txt"/> <file
name="FinalText2.txt"/> -o <file name="FinalText.txt"/></argument>
                <uses name="FinalText1.txt" link="input"/>
                <uses name="FinalText.txt" link="output" register="true"/>
                <uses name="FinalText2.txt" link="input"/>
        </job>
        <child ref="ID0000002">
                <parent ref="ID0000001"/>
        </child>
        <child ref="ID0000003">
                <parent ref="ID0000001"/>
        </child>
        <child ref="ID0000004">
                <parent ref="ID0000002"/>
                <parent ref="ID0000003"/>
        </child>
</adag>
```

4.3.2. Create Site Catalog

In this paper, all the jobs are executed in the condor pool, thus, in Site catalog; site with keyword **condorpool** must be attached. Furthermore, some condor profile also be set, namely, should_transfer_files, when_to_transfer_output, those settings will allow system to transfer all the files that are need for the workflow

```
<sitecatalog xmlns="http://pegasus.isi.edu/schema/sitecatalog"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://pegasus.isi.edu/schema/sitecatalog
http://pegasus.isi.edu/schema/sc-3.0.xsd" version="3.0">
<site handle="condorpool" arch="x86_64" os="LINUX">
<head-fs>
<scratch/>
<storage/>
</head-fs>
<replica-catalog type="LRC" url="rlsn://dummyValue.url.edu"/>
<profile namespace="pegasus" key="style">condor</profile>
<profile namespace="condor" key="universe">vanilla</profile>
<profile namespace="env" key="PEGASUS_HOME">/usr</profile>
<!--
The requirements expression allows you to limit where your jobs go
-->
<!--
The following two profiles forces Condor to always transfer files. This has to be used if the
pool does not have a shared filesystem
-->
<profile namespace="condor" key="should_transfer_files">True</profile>
<profile namespace="condor" key="transfer_output">True</profile>
<profile namespace="condor" key="when_to_transfer_output">ON_EXIT</profile>
<profile namespace="condor" key="requirements">
(Arch==Arch)&&(Disk!=0)&&(Memory!=0)&&(OpSys==OpSys)&&(FileSystemDomain!=
"")
</profile>
</site>
<site handle="local" arch="x86_64" os="LINUX">
<grid type="gt2" contact="localhost/jobmanager-fork" scheduler="Fork"
jobtype="auxillary"/>
<head-fs>
<scratch>
<shared>
<file-server protocol="file" url="file://" mount-point="/home/pega/walkthrough/work"/>
<internal-mount-point mount-point="/home/pega/walkthrough/work"/>
</shared>
</scratch>
<storage>
<shared>
<file-server protocol="file" url="file://" mount-point="/home/pega/walkthrough/storage"/>
<internal-mount-point mount-point="/home/pega/walkthrough/storage"/>
</shared>
</storage>
</head-fs>
<replica-catalog type="LRC" url="rlsn://dummyValue.url.edu"/>
<profile namespace="env" key="PEGASUS_HOME">/usr</profile>
<profile namespace="env" key="GLOBUS_LOCATION">/usr/local/globus-5.2.0</profile>
</site>
</sitecatalog>
```

Figure 24 – Condor Site Catalog

In condorpool environment, all the jobs have temporary directories; all these locations are cleaned up after finishing the workflow. Thus, working locations are not need to be declared.

Furthermore, mount-point="/home/pega/walkthrough/storage" is used as output location that worker node will transfer results to.

4.3.3. Create Pegasus Properties

The following properties also be set

```
pegasus.catalog.site=XML3
pegasus.catalog.site.file=sites.xml
pegasus.data.configuration=condorio
pegasus.transfer.worker.package = False
```

Pegasus with full features is already deployed into image, hence Pegasus worker is abundant, **pegasus.transfer.worker.package** property is set to False. Data staging type here is **condorIO** with the type of catalog site is XML3.

4.3.4. Submit Workflow

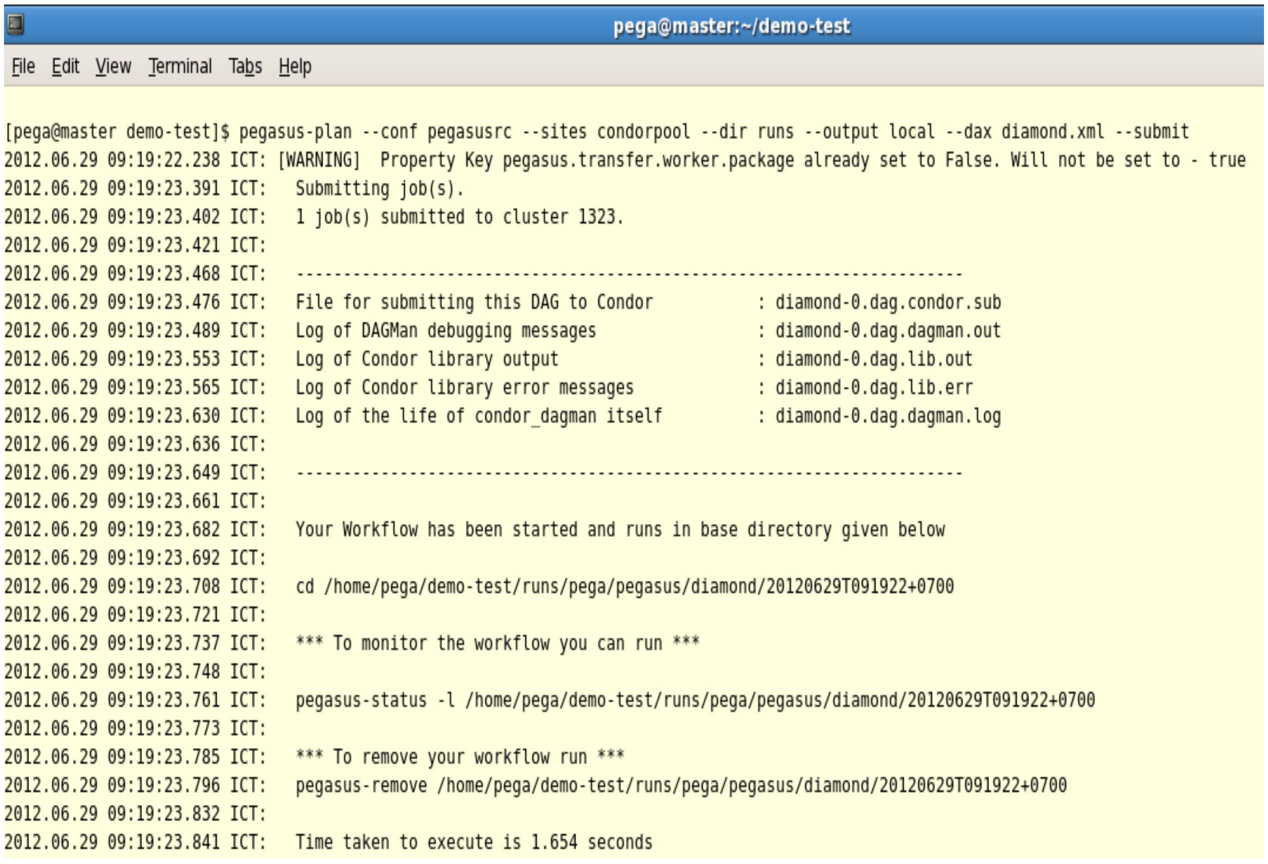
The workflow can be plan and submit via this command

```
pegasus-plan --conf pegasusrc --sites condorpool --dir runs --output local --dax diamond.xml
--submit
```

The workflow is submitted with the following properties

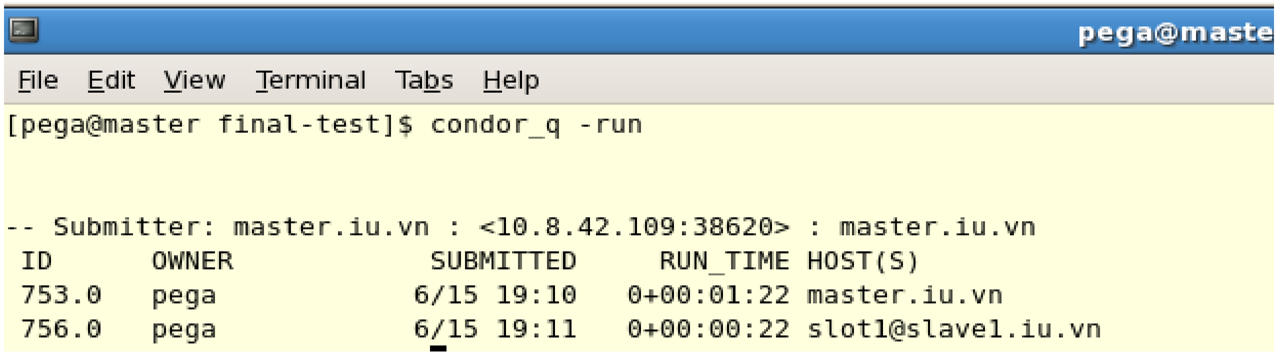
- Pegasus configuration: pegasusrc
- Site to execute jobs: condorpool
- Output location: Local
- Dax file used: diamond.xml

During the process, the job processing can be tracked via Pegasus built-in command, such as, **pegasus-status** with specific working directory; **Pegasus-analyzer** is used in case of error occurrence. **Pegasus-statistic** will shows the statistic of the whole process when the workflow has been finished. In addition, condor_q command also displays info about running job.



```
[pega@master demo-test]$ pegasus-plan --conf pegasusrc --sites condorpool --dir runs --output local --dax diamond.xml --submit
2012.06.29 09:19:22.238 ICT: [WARNING]  Property Key pegasus.transfer.worker.package already set to False. Will not be set to - true
2012.06.29 09:19:23.391 ICT:   Submitting job(s).
2012.06.29 09:19:23.402 ICT:   1 job(s) submitted to cluster 1323.
2012.06.29 09:19:23.421 ICT:
2012.06.29 09:19:23.468 ICT:   -----------------------------------------------------------------------
2012.06.29 09:19:23.476 ICT:   File for submitting this DAG to Condor           : diamond-0.dag.condor.sub
2012.06.29 09:19:23.489 ICT:   Log of DAGMan debugging messages                 : diamond-0.dag.dagman.out
2012.06.29 09:19:23.553 ICT:   Log of Condor library output                     : diamond-0.dag.lib.out
2012.06.29 09:19:23.565 ICT:   Log of Condor library error messages             : diamond-0.dag.lib.err
2012.06.29 09:19:23.630 ICT:   Log of the life of condor_dagman itself          : diamond-0.dag.dagman.log
2012.06.29 09:19:23.636 ICT:
2012.06.29 09:19:23.649 ICT:   -----------------------------------------------------------------------
2012.06.29 09:19:23.661 ICT:
2012.06.29 09:19:23.682 ICT:   Your Workflow has been started and runs in base directory given below
2012.06.29 09:19:23.692 ICT:
2012.06.29 09:19:23.708 ICT:   cd /home/pega/demo-test/runs/pega/pegasus/diamond/20120629T091922+0700
2012.06.29 09:19:23.721 ICT:
2012.06.29 09:19:23.737 ICT:   *** To monitor the workflow you can run ***
2012.06.29 09:19:23.748 ICT:
2012.06.29 09:19:23.761 ICT:   pegasus-status -l /home/pega/demo-test/runs/pega/pegasus/diamond/20120629T091922+0700
2012.06.29 09:19:23.773 ICT:
2012.06.29 09:19:23.785 ICT:   *** To remove your workflow run ***
2012.06.29 09:19:23.796 ICT:   pegasus-remove /home/pega/demo-test/runs/pega/pegasus/diamond/20120629T091922+0700
2012.06.29 09:19:23.832 ICT:
2012.06.29 09:19:23.841 ICT:   Time taken to execute is 1.654 seconds
```

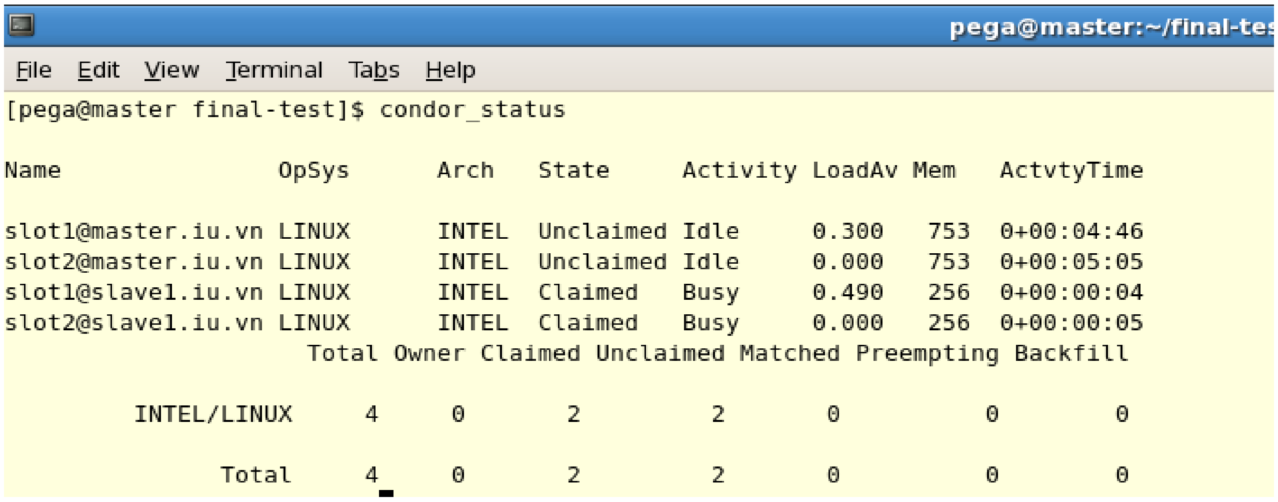
Figure 25 – Submit workflow



```
[pega@master final-test]$ condor_q -run


-- Submitter: master.iu.vn : <10.8.42.109:38620> : master.iu.vn
 ID      OWNER            SUBMITTED     RUN_TIME HOST(S)
 753.0   pega            6/15 19:10   0+00:01:22 master.iu.vn
 756.0   pega            6/15 19:11   0+00:00:22 slot1@slave1.iu.vn
```

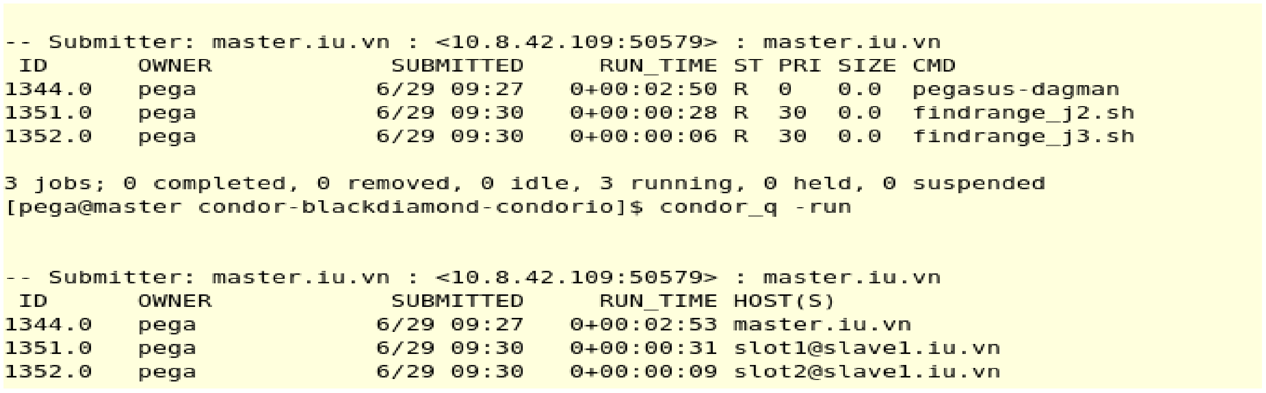
Figure 26 – Workflow running on worker node 1

In case two jobs run concurrently, two resources are displayed as claimed as below figure.



```
[pega@master final-test]$ condor_status

Name               OpSys      Arch   State     Activity LoadAv Mem   ActvtyTime

slot1@master.iu.vn LINUX      INTEL  Unclaimed Idle     0.300   753  0+00:04:46
slot2@master.iu.vn LINUX      INTEL  Unclaimed Idle     0.000   753  0+00:05:05
slot1@slave1.iu.vn LINUX      INTEL  Claimed   Busy     0.490   256  0+00:00:04
slot2@slave1.iu.vn LINUX      INTEL  Claimed   Busy     0.000   256  0+00:00:05
                     Total Owner Claimed Unclaimed Matched Preempting Backfill

         INTEL/LINUX     4     0       2         2       0          0        0

               Total     4     0       2         2       0          0        0
```

Figure 27 – Resources claimed

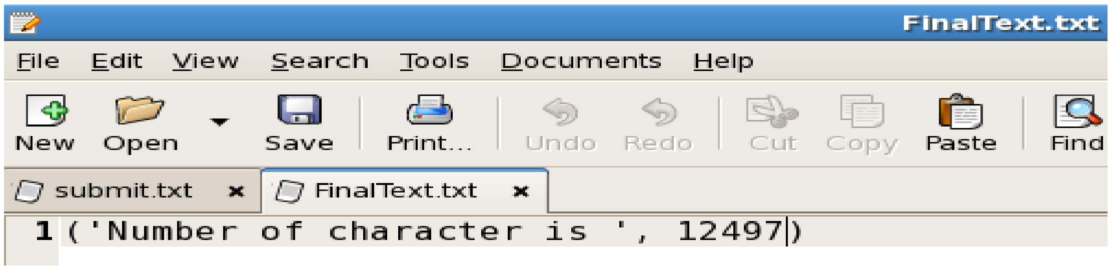


```
-- Submitter: master.iu.vn : <10.8.42.109:50579> : master.iu.vn
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
1344.0   pega            6/29 09:27   0+00:02:50 R  0   0.0  pegasus-dagman
1351.0   pega            6/29 09:30   0+00:00:28 R  30  0.0  findrange_j2.sh
1352.0   pega            6/29 09:30   0+00:00:06 R  30  0.0  findrange_j3.sh

3 jobs; 0 completed, 0 removed, 0 idle, 3 running, 0 held, 0 suspended
[pega@master condor-blackdiamond-condorio]$ condor_q -run


-- Submitter: master.iu.vn : <10.8.42.109:50579> : master.iu.vn
 ID      OWNER            SUBMITTED     RUN_TIME HOST(S)
1344.0   pega            6/29 09:27   0+00:02:53 master.iu.vn
1351.0   pega            6/29 09:30   0+00:00:31 slot1@slave1.iu.vn
1352.0   pega            6/29 09:30   0+00:00:09 slot2@slave1.iu.vn
```

Figure 28 – Concurrently running Job

The condor_q command shows which job is running at the moment.

The final output is FinalText.txt which contains the total number character of input file text.txt.



```
('Number of character is ', 12497)
```

Figure 29 – Output file

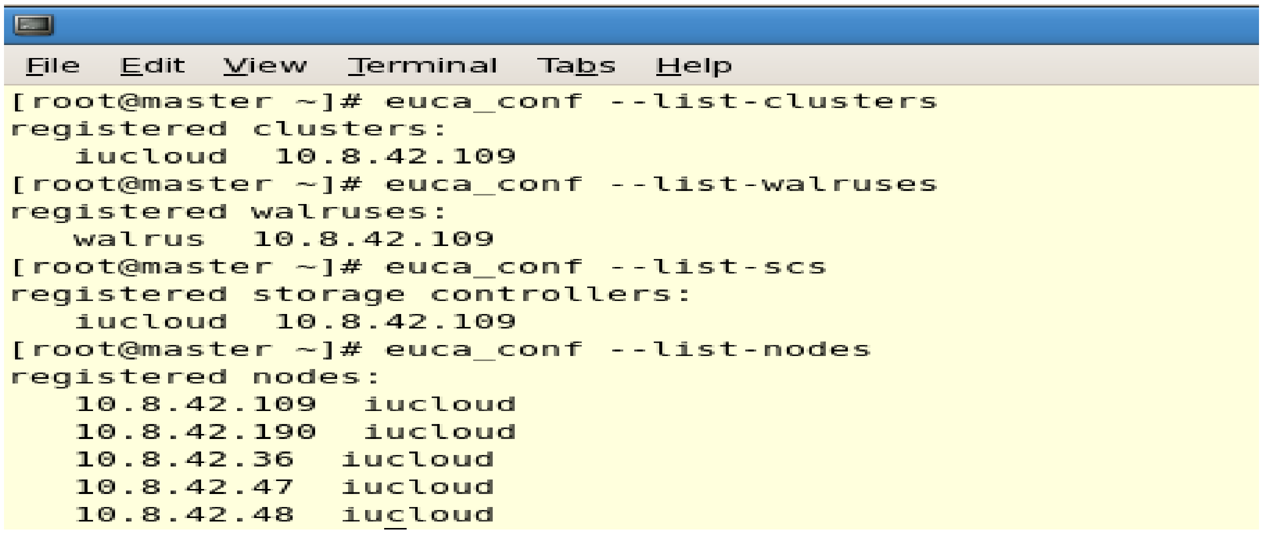
CHAPTER 5

EVALUATION

Two main points will be assessed in this section, namely, the infrastructure of system and workflow processing.

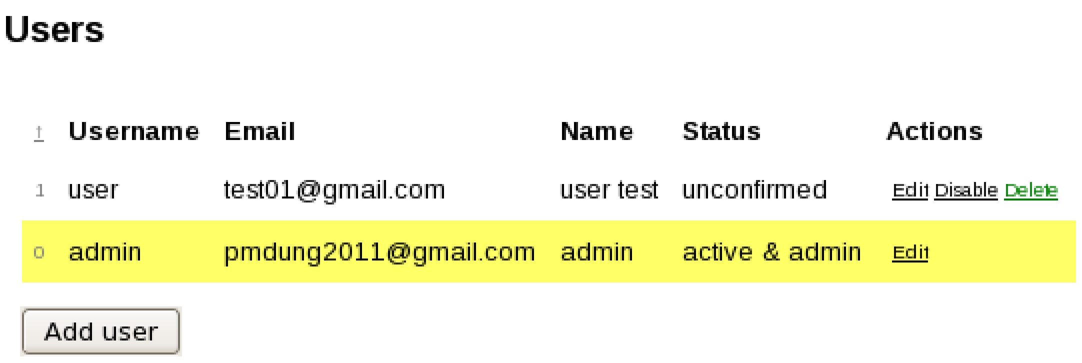
5.1 System Infrastructure

At the first point, the system is developed on open-source cloud platform, Eucalyptus software has been chosen to build up this system. Eucalyptus provides powerful features with various benefits to build a private cloud platform. In this work, Eucalyptus cost lots of time to build from bottom to top, require the administrators should have special skills in both networking and Linux Operating System, fortunately, with my attempt, the open-source Eucalyptus cloud platform has been built successfully. The following features are used to demonstrate all the functions of Eucalyptus in my system.



```
[root@master ~]# euca_conf --list-clusters
registered clusters:
   iucloud  10.8.42.109
[root@master ~]# euca_conf --list-walruses
registered walruses:
   walrus  10.8.42.109
[root@master ~]# euca_conf --list-scs
registered storage controllers:
   iucloud  10.8.42.109
[root@master ~]# euca_conf --list-nodes
registered nodes:
   10.8.42.109   iucloud
   10.8.42.190   iucloud
   10.8.42.36   iucloud
   10.8.42.47   iucloud
   10.8.42.48   iucloud
```

Figure 30 – Registered Components



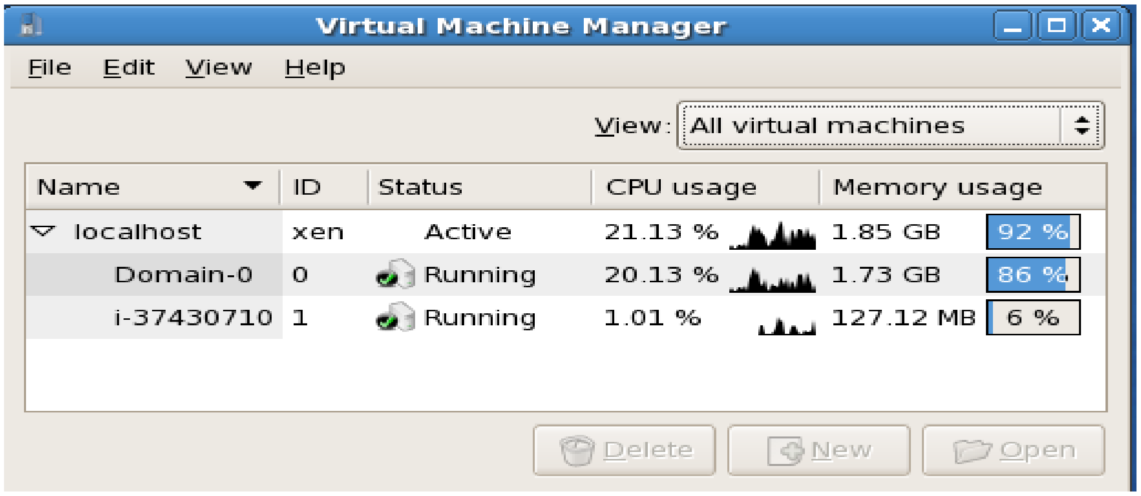
Users

↕	Username	Email	Name	Status	Actions
1	user	test01@gmail.com	user test	unconfirmed	Edit Disable Delete
0	admin	pmdung2011@gmail.com	admin	active & admin	Edit

Figure 31 – User Information

All components are successfully registered with correct information.

Due to the limit of equipment, the power of cloud platform could not be considered in good manner. My laptop has been used to work as a front-end, with only two Gigabyte of memory; only one large type instance can be started up, similarly with two small type instances.



Name	ID	Status	CPU usage	Memory usage
localhost	xen	Active	21.13 %	1.85 GB 92 %
Domain-0	0	Running	20.13 %	1.73 GB 86 %
i-37430710	1	Running	1.01 %	127.12 MB 6 %

Figure 32 – CPU usage

In addition, startup time of virtual machines is heavily depended on the network status. Before running a virtual machine, the cloud runs a few steps:

1. Check availability of the cloud to check if there is enough resource.
2. Decrypt and decompress the image
3. At the same time, send the image to one of the node.
4. The node will save the image to the cache.
5. And copy a new copy and run the virtual machine. The virtualization used

Size of the image	First start up time
3Gb	~5mins
5Gb	~8mins
10Gb	~17mins

Figure 33 – Startup time of different instance type

When an image is sent to the node server, it will copy and save if node has sufficient space. Next time if the same image runs on it, it can directly copy save the bandwidth. Second start up requires 1/3 of the first start up time only.

5.2 Workflow Processing

5.2.1 Built-in Application

For precise evaluation I use a Pegasus-keg, which is built-in executable to solve diamond-shaped workflow follows *Figure-9* as a validation for this work. The workflow is temporary called as Condor Workflow. Key feature of Pegasus-keg is that it can record the hostname of the host it ran upon, the current timestamp, the run time from start, the current working directory and some information on the system environment also. In this example, the input file "f.a" is used as the blank input; all the jobs are processed by Pegasus-keg, at the result all the information about the jobs such as specific executable applications, runtime will be recorded in output file.

In order to prove the validation of various DAX generators, Python is not used but Java is the programming languages used to generate abstract workflow. The code used is attached in built-in Pegasus-keg application.

```
import edu.isi.pegasus.planner.dax.*;

public class BlackDiamondDAX {

   /**
    * Create an example DIAMOND DAX
    * @param args
    */
   public static void main(String[] args) {
      if (args.length != 2) {
         System.out.println("Usage: java ADAG <pegasus_bin_dir> <filename.dax>");
         System.exit(1);
      }

      try {
         Diamond(args[0]).writeToFile(args[1]);
      }
      catch (Exception e) {
         e.printStackTrace();
      }

   }

   private static ADAG Diamond(String pegasus_bin_dir) throws Exception {

      java.io.File cwdFile = new java.io.File (".");
      String cwd = cwdFile.getCanonicalPath();

      ADAG dax = new ADAG("blackdiamond");

      File fa = new File("f.a");
      fa.addPhysicalFile("file://" + cwd + "/f.a", "local");
      dax.addFile(fa);

      File fb1 = new File("f.b1");
      File fb2 = new File("f.b2");
      File fc1 = new File("f.c1");
      File fc2 = new File("f.c2");
      File fd = new File("f.d");
      fd.setRegister(true);
```

```
Executable preprocess = new Executable("pegasus", "preprocess", "4.0");
preprocess.setArchitecture(Executable.ARCH.X86).setOS(Executable.OS.LINUX);
preprocess.setInstalled(false);
preprocess.addPhysicalFile("file://" + pegasus_bin_dir + "/pegasus-keg", "local");

Executable findrange = new Executable("pegasus", "findrange", "4.0");
findrange.setArchitecture(Executable.ARCH.X86).setOS(Executable.OS.LINUX);
findrange.setInstalled(false);
findrange.addPhysicalFile("file://" + pegasus_bin_dir + "/pegasus-keg", "local");

Executable analyze = new Executable("pegasus", "analyze", "4.0");
analyze.setArchitecture(Executable.ARCH.X86).setOS(Executable.OS.LINUX);
analyze.setInstalled(false);
analyze.addPhysicalFile("file://" + pegasus_bin_dir + "/pegasus-keg", "local");

dax.addExecutable(preprocess).addExecutable(findrange).addExecutable(analyze);

// Add a preprocess job
Job j1 = new Job("j1", "pegasus", "preprocess", "4.0");
j1.addArgument("-a preprocess -T 60 -i ").addArgument(fa);
j1.addArgument("-o ").addArgument(fb1);
j1.addArgument(" ").addArgument(fb2);
j1.uses(fa, File.LINK.INPUT);
j1.uses(fb1, File.LINK.OUTPUT);
j1.uses(fb2, File.LINK.OUTPUT);
dax.addJob(j1);

// Add left Findrange job
Job j2 = new Job("j2", "pegasus", "findrange", "4.0");
j2.addArgument("-a findrange -T 60 -i ").addArgument(fb1);
j2.addArgument("-o ").addArgument(fc1);
j2.uses(fb1, File.LINK.INPUT);
j2.uses(fc1, File.LINK.OUTPUT);
dax.addJob(j2);

// Add right Findrange job
Job j3 = new Job("j3", "pegasus", "findrange", "4.0");
j3.addArgument("-a findrange -T 60 -i ").addArgument(fb2);
j3.addArgument("-o ").addArgument(fc2);
j3.uses(fb2, File.LINK.INPUT);
j3.uses(fc2, File.LINK.OUTPUT);
dax.addJob(j3);

// Add analyze job
Job j4 = new Job("j4", "pegasus", "analyze", "4.0");
j4.addArgument("-a analyze -T 60 -i ").addArgument(fc1);
```

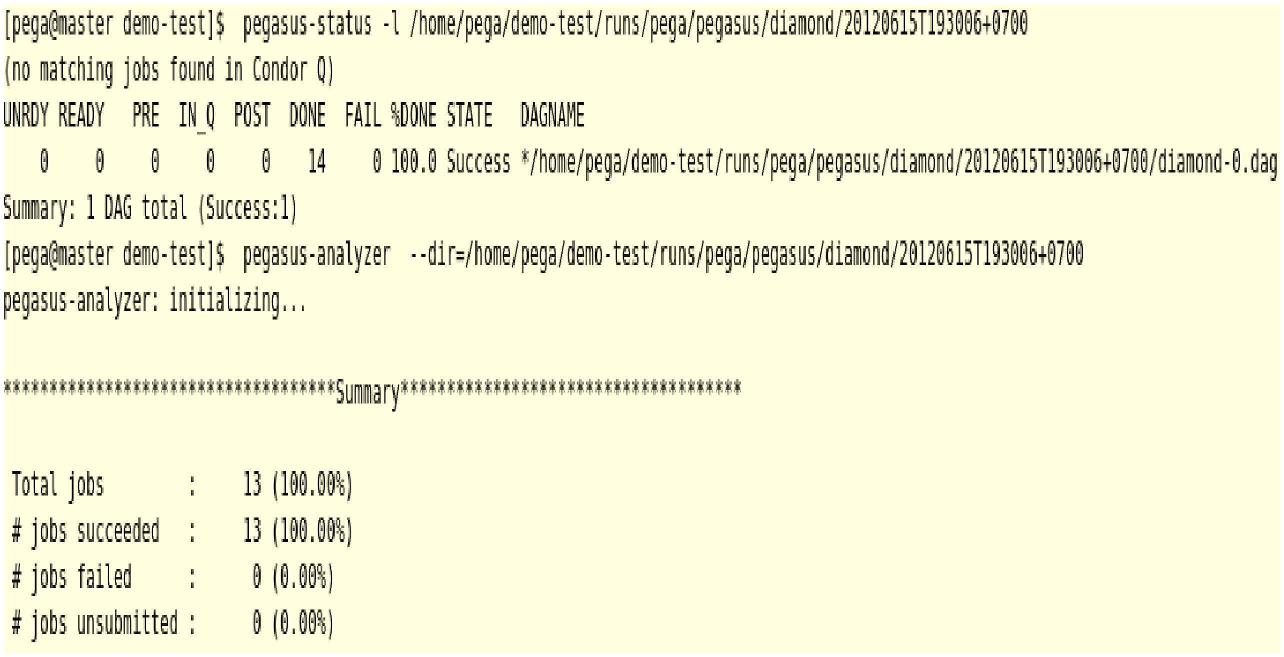
```
      j4.addArgument(" ").addArgument(fc2);
      j4.addArgument("-o ").addArgument(fd);
      j4.uses(fc1, File.LINK.INPUT);
      j4.uses(fc2, File.LINK.INPUT);
      j4.uses(fd, File.LINK.OUTPUT);
      dax.addJob(j4);

      dax.addDependency("j1", "j2");
      dax.addDependency("j1", "j3");
      dax.addDependency("j2", "j4");
      dax.addDependency("j3", "j4");
      return dax;
   }
}
```

A workflow with 13 jobs can be executed within 143 seconds. The whole process can be tracked and monitored via built-in Pegasus commands. The five outputs of this workflow are f.b1, f.b2, f.c1, f.c2 and f.d. Each file displays the information such as below, with f.b1

```
--- start f.a ----
  This is sample input to KEG
--- final f.a ----
Timestamp Today: 20120624T062423.066-04:00 (1340533463.066;60.000)
Applicationname: preprocess [v5072] @ 10.8.42.111 (VPN)
Current Workdir: /var/lib/condor/execute/dir_4450
Systemenvironm.: i686-Linux 2.6.24-19-xen
Processor Info.: 2 x Intel(R) Core(TM)2 Duo CPU     P8400  @ 2.26GHz @
2261.256
Load Averages  : 3.630 3.298 3.176
Memory Usage MB: 256 total, 4 free, 0 shared, 47 buffered
Swap Usage   MB: 0 total, 0 free
Filesystem Info: /                        ext3  2519MB total,   707MB
avail
Output Filename: f.b1
Input Filenames: f.a
```

With this information, the workflow processing is validated. For example, the application used to execute this file is “preprocess” and temporary working directory. The Output and Input file name also be showed. The other outputs have similar information.

All the specific jobs listed out with detail information when the workflow has finished. Only clicking on specific area that we want to know, detail information contained in simple box will shows out. During the process, user can also track or monitor the submitted workflow. In case of errors occur, notification also be listed out with detail information.



```
[pega@master demo-test]$  pegasus-status -l /home/pega/demo-test/runs/pega/pegasus/diamond/20120615T193006+0700
(no matching jobs found in Condor Q)
UNRDY READY   PRE  IN_Q  POST  DONE  FAIL %DONE STATE   DAGNAME
    0     0     0     0     0    14     0 100.0 Success */home/pega/demo-test/runs/pega/pegasus/diamond/20120615T193006+0700/diamond-0.dag
Summary: 1 DAG total (Success:1)
[pega@master demo-test]$  pegasus-analyzer  --dir=/home/pega/demo-test/runs/pega/pegasus/diamond/20120615T193006+0700
pegasus-analyzer: initializing...

************************************Summary*************************************

 Total jobs         :     13 (100.00%)
 # jobs succeeded   :     13 (100.00%)
 # jobs failed      :      0 (0.00%)
 # jobs unsubmitted :      0 (0.00%)
```

Figure 34–Job tracking

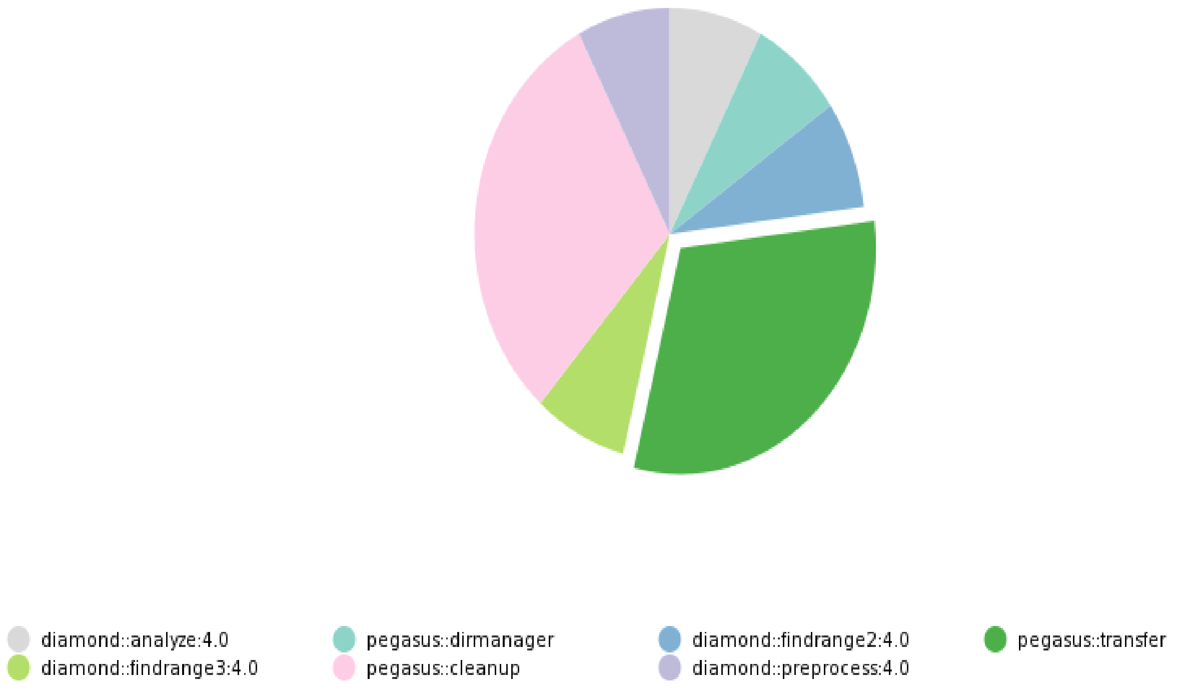
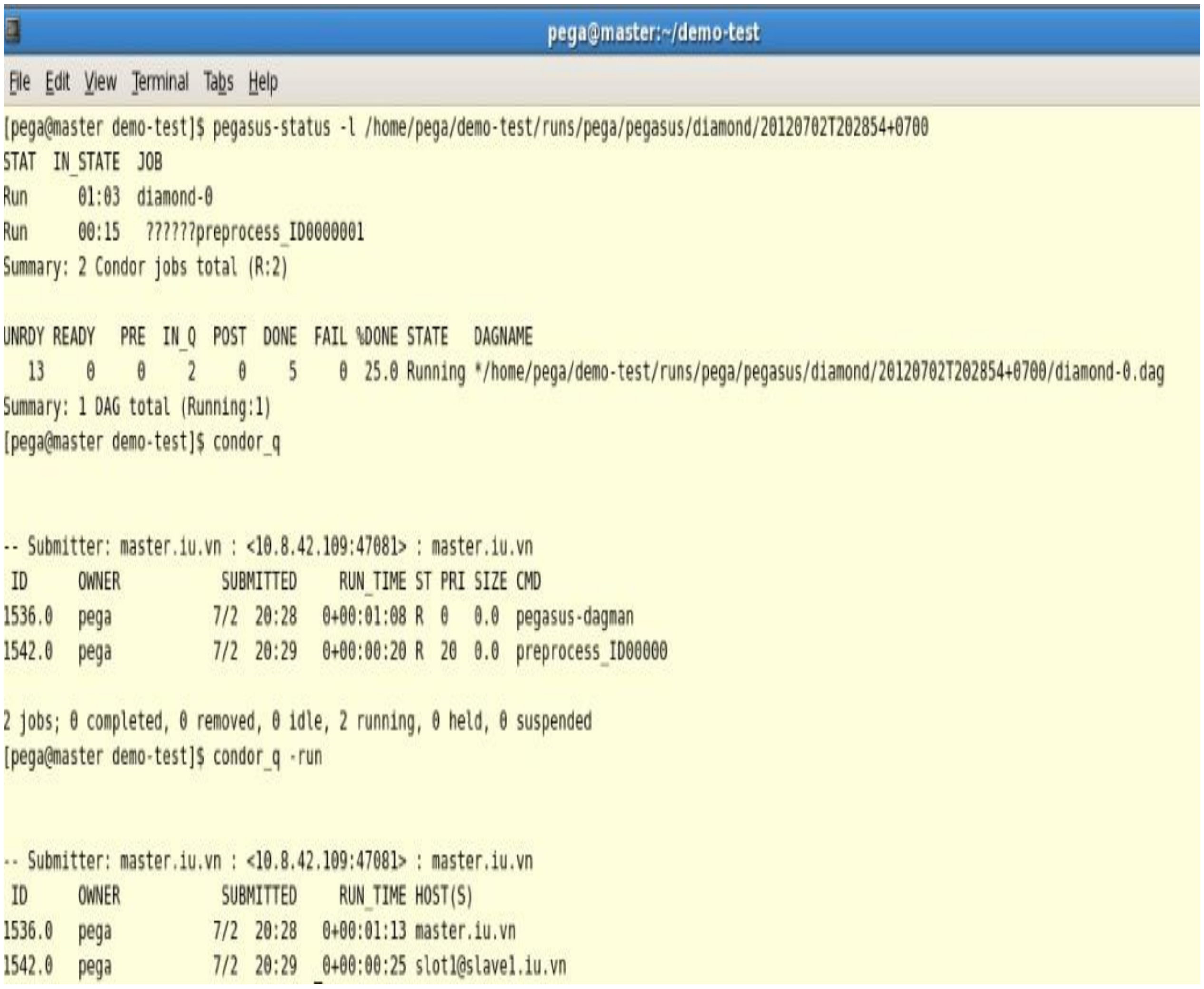


Figure 35 – Condor Workflow Job Diagram

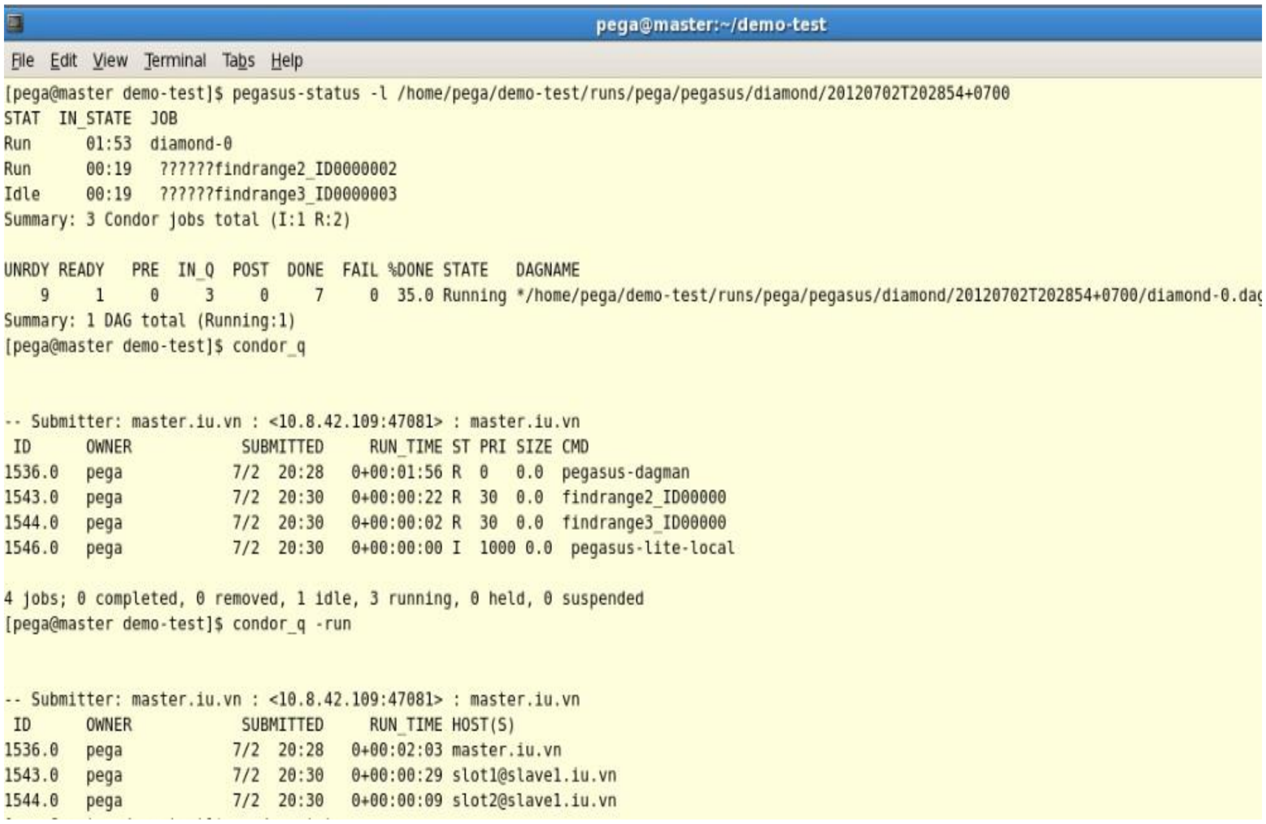
5.2.2 CharacterCount Workflow

With CharacterCount Workflow, there are 4 main tasks talked above, and 20 Jobs. The total runtime is 2 mins 9 secs, (total 129 seconds). With the four main tasks, the preprocess-Job1 took only 0.035s; findrange2-Job2 and findrange3-Job3 took 0.01s, analyze-Job4 run a bit slower than Job2 and Job3 with 0.011s. Furthermore, all jobs are executed with correct order following submitted abstract workflow, namely, the first job is executed firstly, after that, the next two jobs are concurrently performed. Finally, the job4 then be taken. In addition, using **Pegasus-plots** command, a detail diagram with runtime in second of the workflow will be displayed, which is similar to Condor Workflow.



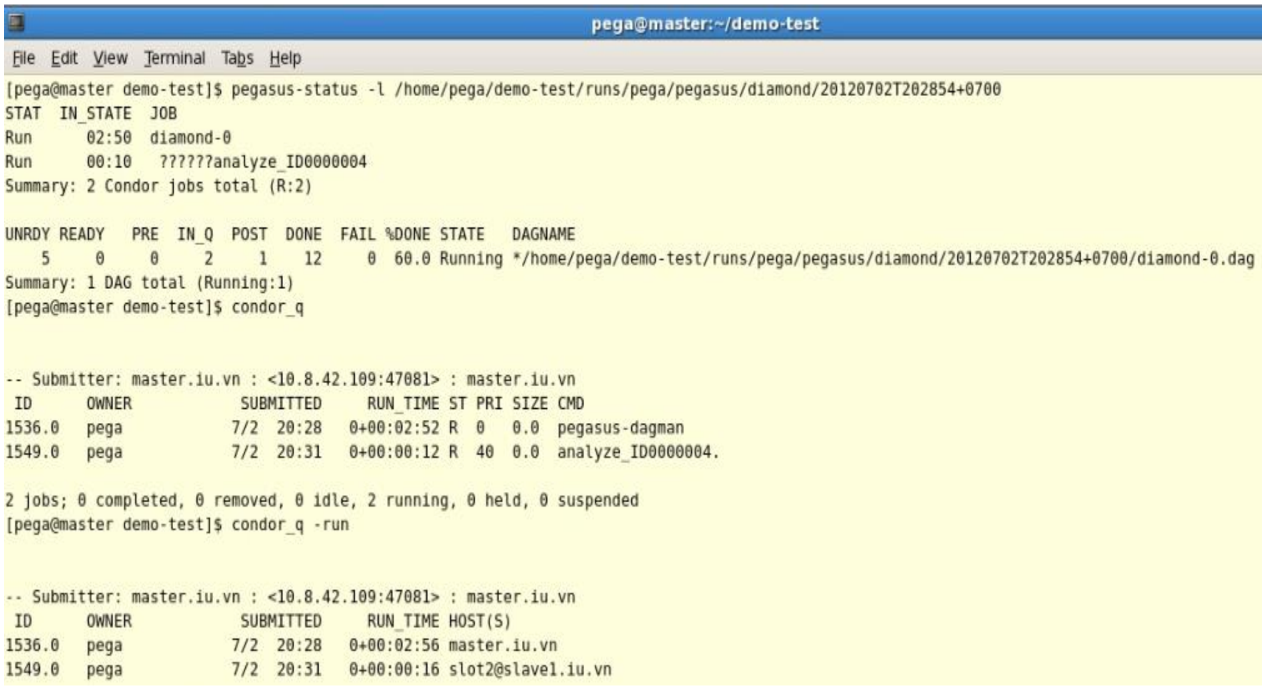
```
[pega@master demo-test]$ pegasus-status -l /home/pega/demo-test/runs/pega/pegasus/diamond/20120702T202854+0700
STAT  IN_STATE  JOB
Run      01:03  diamond-0
Run      00:15   ??????preprocess_ID0000001
Summary: 2 Condor jobs total (R:2)

UNRDY READY   PRE  IN_Q  POST  DONE  FAIL %DONE STATE   DAGNAME
   13     0     0     2     0     5     0  25.0 Running */home/pega/demo-test/runs/pega/pegasus/diamond/20120702T202854+0700/diamond-0.dag
Summary: 1 DAG total (Running:1)
[pega@master demo-test]$ condor_q


-- Submitter: master.iu.vn : <10.8.42.109:47081> : master.iu.vn
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
1536.0   pega            7/2  20:28   0+00:01:08 R  0   0.0  pegasus-dagman
1542.0   pega            7/2  20:29   0+00:00:20 R  20  0.0  preprocess_ID00000

2 jobs; 0 completed, 0 removed, 0 idle, 2 running, 0 held, 0 suspended
[pega@master demo-test]$ condor_q -run


-- Submitter: master.iu.vn : <10.8.42.109:47081> : master.iu.vn
 ID      OWNER            SUBMITTED     RUN_TIME HOST(S)
1536.0   pega            7/2  20:28   0+00:01:13 master.iu.vn
1542.0   pega            7/2  20:29   0+00:00:25 slot1@slave1.iu.vn
```

Figure 36 – Job1 is executed firstly



```
[pega@master demo-test]$ pegasus-status -l /home/pega/demo-test/runs/pega/pegasus/diamond/20120702T202854+0700
STAT  IN_STATE  JOB
Run      01:53  diamond-0
Run      00:19   ??????findrange2_ID0000002
Idle     00:19   ??????findrange3_ID0000003
Summary: 3 Condor jobs total (I:1 R:2)

UNRDY READY   PRE  IN_Q  POST  DONE  FAIL %DONE STATE   DAGNAME
    9     1     0     3     0     7     0  35.0 Running */home/pega/demo-test/runs/pega/pegasus/diamond/20120702T202854+0700/diamond-0.dag
Summary: 1 DAG total (Running:1)
[pega@master demo-test]$ condor_q


-- Submitter: master.iu.vn : <10.8.42.109:47081> : master.iu.vn
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
1536.0   pega            7/2  20:28   0+00:01:56 R  0   0.0  pegasus-dagman
1543.0   pega            7/2  20:30   0+00:00:22 R  30  0.0  findrange2_ID00000
1544.0   pega            7/2  20:30   0+00:00:02 R  30  0.0  findrange3_ID00000
1546.0   pega            7/2  20:30   0+00:00:00 I  1000 0.0  pegasus-lite-local

4 jobs; 0 completed, 0 removed, 1 idle, 3 running, 0 held, 0 suspended
[pega@master demo-test]$ condor_q -run


-- Submitter: master.iu.vn : <10.8.42.109:47081> : master.iu.vn
 ID      OWNER            SUBMITTED     RUN_TIME HOST(S)
1536.0   pega            7/2  20:28   0+00:02:03 master.iu.vn
1543.0   pega            7/2  20:30   0+00:00:29 slot1@slave1.iu.vn
1544.0   pega            7/2  20:30   0+00:00:09 slot2@slave1.iu.vn
```

Figure 37 – Job2 and Job3 are concurrently executed



```
[pega@master demo-test]$ pegasus-status -l /home/pega/demo-test/runs/pega/pegasus/diamond/20120702T202854+0700
STAT  IN_STATE  JOB
Run      02:50  diamond-0
Run      00:10   ??????analyze_ID0000004
Summary: 2 Condor jobs total (R:2)

UNRDY READY   PRE  IN_Q  POST  DONE  FAIL %DONE STATE   DAGNAME
    5     0     0     2     1    12     0  60.0 Running */home/pega/demo-test/runs/pega/pegasus/diamond/20120702T202854+0700/diamond-0.dag
Summary: 1 DAG total (Running:1)
[pega@master demo-test]$ condor_q


-- Submitter: master.iu.vn : <10.8.42.109:47081> : master.iu.vn
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
1536.0   pega            7/2  20:28   0+00:02:52 R  0   0.0  pegasus-dagman
1549.0   pega            7/2  20:31   0+00:00:12 R  40  0.0  analyze_ID0000004.

2 jobs; 0 completed, 0 removed, 0 idle, 2 running, 0 held, 0 suspended
[pega@master demo-test]$ condor_q -run


-- Submitter: master.iu.vn : <10.8.42.109:47081> : master.iu.vn
 ID      OWNER            SUBMITTED     RUN_TIME HOST(S)
1536.0   pega            7/2  20:28   0+00:02:56 master.iu.vn
1549.0   pega            7/2  20:31   0+00:00:16 slot2@slave1.iu.vn
```

Figure 38 – Job4 is performed at last

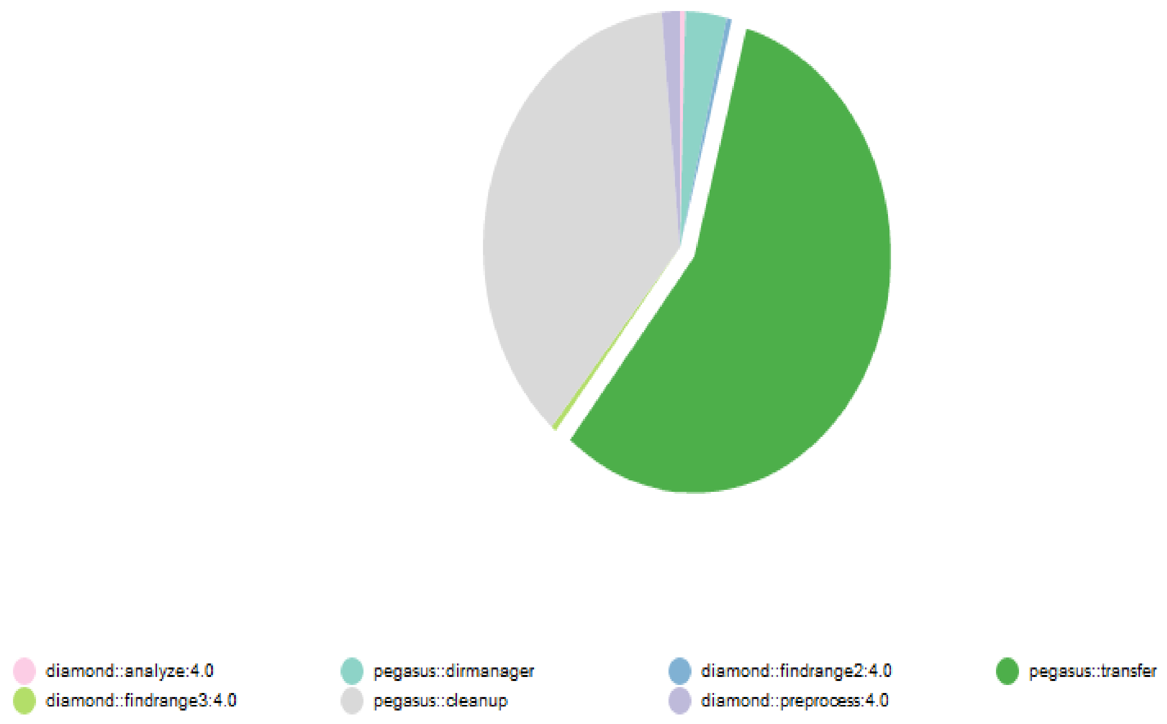
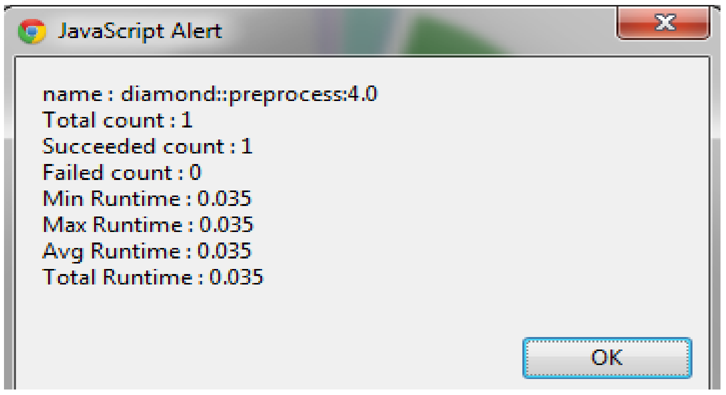
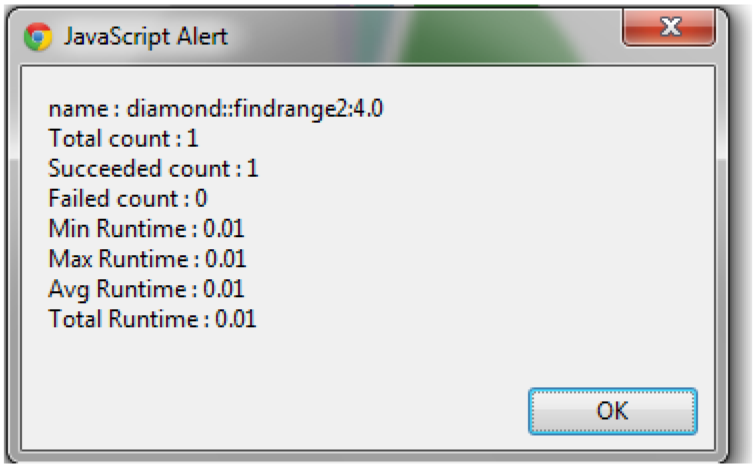


Figure 39 – CharacterCount Workflow Runtime Diagram



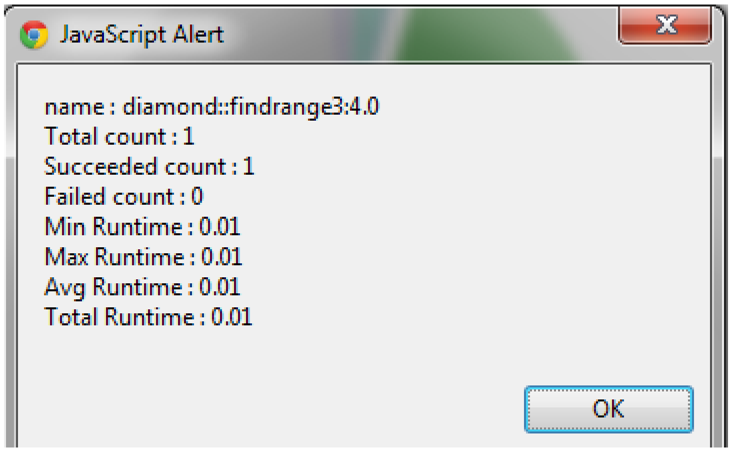
name : diamond::preprocess:4.0
Total count : 1
Succeeded count : 1
Failed count : 0
Min Runtime : 0.035
Max Runtime : 0.035
Avg Runtime : 0.035
Total Runtime : 0.035

Figure 40 – Job1 Runtime



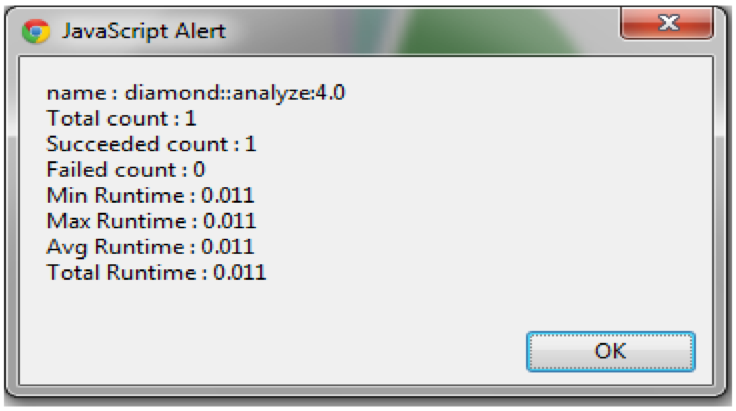
```
name : diamond::findrange2:4.0
Total count : 1
Succeeded count : 1
Failed count : 0
Min Runtime : 0.01
Max Runtime : 0.01
Avg Runtime : 0.01
Total Runtime : 0.01
```

Figure 41 – Job2 Runtime



```
name : diamond::findrange3:4.0
Total count : 1
Succeeded count : 1
Failed count : 0
Min Runtime : 0.01
Max Runtime : 0.01
Avg Runtime : 0.01
Total Runtime : 0.01
```

Figure 42 – Job3 Runtime



```
name : diamond::analyze:4.0
Total count : 1
Succeeded count : 1
Failed count : 0
Min Runtime : 0.011
Max Runtime : 0.011
Avg Runtime : 0.011
Total Runtime : 0.011
```

Figure 43 – Job4 Runtime

For each finished workflow, respectively log file would be created. For more particular information about the job processing, the log file should be considered. Some part of the log file of CharacterCount Workflow is taken as an example.

```
...
000 (1329.000.000) 06/29 09:20:10 Job submitted from host:
<10.8.42.109:50579>
    DAG Node: preprocess_ID0000001
    pool:condorpool
...
001 (1329.000.000) 06/29 09:20:23 Job executing on host:
<10.8.42.110:48773>
...
006 (1329.000.000) 06/29 09:20:24 Image size of job updated: 1
     0  -  MemoryUsage of job (MB)
     0  -  ResidentSetSize of job (KB)
...
005 (1329.000.000) 06/29 09:20:24 Job terminated.
     (1) Normal termination (return value 0)
          Usr 0 00:00:00, Sys 0 00:00:00  -  Run Remote Usage
          Usr 0 00:00:00, Sys 0 00:00:00  -  Run Local Usage
          Usr 0 00:00:00, Sys 0 00:00:00  -  Total Remote Usage
          Usr 0 00:00:00, Sys 0 00:00:00  -  Total Local Usage
     8736  -  Run Bytes Sent By Job
     9917  -  Run Bytes Received By Job
     8736  -  Total Bytes Sent By Job
     9917  -  Total Bytes Received By Job
     Partitionable Resources :    Usage  Request
        Cpus                 :               1
        Disk (KB)            :       27     27
        Memory (MB)          :        0      0
...
```

With the first job, Preprocess, is submitted from host with Ip address is 10.8.42.109, then the job is executed on one worker node has Ip 10.8.42.110. All the jobs are executed in condorpool environment. To execute this job, 27 KyloByte of resources has claimed. Similarly, job2 and job3 has the same information included in log file.

```
...
000 (1330.000.000) 06/29 09:20:36 Job submitted from host:
<10.8.42.109:50579>
    DAG Node: findrange2_ID0000002
    pool:condorpool
...
000 (1331.000.000) 06/29 09:20:36 Job submitted from host:
<10.8.42.109:50579>
    DAG Node: findrange3_ID0000003
    pool:condorpool
...
001 (1330.000.000) 06/29 09:20:37 Job executing on host:
<10.8.42.110:48773>
...
```

The limit of this system is that it does not include Globus tools; an open source software toolkit used for building grids, there is due to the restriction of time and insufficient hardware also, I could not successfully deploy the software into the system, and hence the features of the whole system are restricted within condor pool of VMs. Furthermore, huge applications have not been deployed, thus, the power of this system might be underestimated. However, the rest functions are properly work, all components and services work correctly and stably.

CHAPTER 6

CONCLUSION AND FUTURE WORK

Cloud computing is undoubted to be a powerful IT technology. Its elastic property allows application resizing itself to any scale. The almost import benefits are that cloud offers not only computing service but also various services such as PaaS, IaaS, …Furthermore, the pay-on-demand characteristic of cloud enable small company or organization like school running large scale application or experiment with a reasonable cost.

Even though the cloud provides users with elastic resource, not all application can directly run on cloud. To fully shows up the performance of cloud. Second approach requires some powerful framework on top of all virtual machines and fully used up all resource form each virtual machine.

The system built with Pegasus and Condor, namely is Workflow Management System (WMS) is one of the very powerful computing systems which are suitable to build on cloud platform. Its objective is help building a computing system with high scalability with thousands of virtual machines offering by cloud platform. While Pegasus, Condor implemented in system makes applications run in parallel, it also provide monitoring features, in addition, if any job crash, the job could be recovered by other machine, and will take up the jobs automatically. We can see that developing a computing service on cloud platform is best solution replacing for gird platform in the future.

However, both WMS and Eucalyptus are open-source software and not mature application. There is not enough sufficient document and instruction, they are still being developed. Thus, bugs and issues are unavoidable. Procedure of building the system is very complicated and require developers a huge attempt. Besides that, the lack of hardware resource also a significant problem with the cloud owner, with inadequate hardware resource, the power of cloud platform could not be considered.

In this thesis program, the User Graphic Interface for job-submitter has not implemented yet. I have done lots of effort on building cloud environment and configuring workflow management system, writing applications to learn the characteristic of Workflow system and cloud platform also. In the future, building a Workflow System intend to normal end-user will be focused; the system will not require any special knowledge from end-user. The large workflow also be attempted to run on system will also be done in the future.

LIST OF ACRONYMS

AWS	**Amazon Web Services**
CC	Cluster Controller
CLC	Cloud Controller
CSS	Cascading Style Sheets
CaaS	Computing as a Services
DAGMan	Directed Acyclic Graph Manager
EBS	Elastic Block Storage
HTML	Hyper Text Markup Language
HTC	High Throughput Computing
IaaS	Infrastructure as a Services
LAN	Local Area Network
NCs	Node Controller
PaaS	Platform as a Services
SSH	Secure Shell
SC	Storage Controller
SaaS	Software as a Services
VMs	Virtual Machine
WMS	Workflow Management System
Web UI	Web User Interface
XML	eXtensible Markup Language

REFERENCES

[1] Cloud Computing and Science – Mathematica Embraces Cloud Computing. Retrived from http://www.cloudave.com/2711/cloud-computing-and-science-mathematica-on-clouds/

[2] Workflow. In Wikipedia. Retrieved June 25, 2012, from http://en.wikipedia.org/wiki/Workflow

[3] Judy, Q. Adam, L. H. (n.d). Parallel Applications And Tools For Cloud Computing Environments

[4] Pegasus. (2012 June). Retrieved from http://pegasus.isi.edu/

[5] Condor High Throughput Computing. (2012 June). Retrieved from http://research.cs.wisc.edu/condor/manual/v7.8/

[6] DAGMan. (2012, June). Retrieved from http://research.cs.wisc.edu/condor/dagman/

[7] Eucalyptus. (2012, June). Retrieved from http://www.eucalyptus.com/participate

[8]Daniel Nurmi, R. W. (n.d.). “The Eucalyptus Open-source Cloud-computing System.” 1-5.

[9]D. Nurmi et al., “The Eucalyptus Open-source Cloud-computing System,” IEEE International Symposium on Cluster Computing and the Grid (CCGrid '09), 2009.

[10] Mike, T. Karim, D. (2009). Cloud Computing and its application to Image Processing

[11] Dimitri, B. Sergio, E. (n.d). Grid computing for energy exploration

[12] Grid Computing. (2012, June). Retrieved from Wikipedia http://en.wikipedia.org/wiki/Grid_computing

[13] Virtualization. (2012, June). Retrieved from Wikipedia http://en.wikipedia.org/wiki/Virtualization

[14] Elastic Compute Cloud (EC2). (2012, June). Retrieved from amazon.com http://aws.amazon.com/ec2.

[15] Open-Source Business Models Aren't Dead-End Streets. (2009, November). Retrieved from http://gigaom.com/2009/11/30/open-source-business-models-arent-dead-end-streets/

[16]OpenNebula. (2012, June). Retrieved from http://www.opennebula.org.

[17] Nimbus. (2012, June). Retrieved from http://workspace.globus.org.

[18] VMWare. (2012, June). Retrieved from http://en.wikipedia.org/wiki/VMware

[19] XEN. (2012, June). Retrieved from Wikipedia http://en.wikipedia.org/wiki/Xen

[20] Xen, KVM and the Linux Choice. (2011, July). Retrieved from http://virtualization.sys-con.com/node/1900898

[21] Cloud Computing (2012, June). Retrieved from Wikipedia http://en.wikipedia.org/wiki/Cloud_computing

[22] RFC 3720 - Internet Small Computer Systems Interface *(*iSCSI*)*. In IETF. Retrieved from http://www.ietf.org/rfc/rfc3720.txt

[23] ATA over Ethernet. (2012, June). Retrieved from Wikipedia http://en.wikipedia.org/wiki/ATA_over_Ethernet

[24] Karan, V. Kent, W. (2011). Pegasus WMS Tutorial.

[25] A DAX, a DAG and the work to be done. (2011). Retrieved from http://www.ligo.caltech.edu/~bdaudert/INSPIRAL/DAG_DAX_and_Pegasus.pdf

[26] High Throughput Computing. (2012, June). Retrieved from Wikipedia http://en.wikipedia.org/wiki/High-throughput_computing

[27] Ewa, D. Miron, L. Gaurang, M. Andrew, P. (2009) Pegasus and DAGMan From Concept to Execution: Mapping Scientific Workflows onto Today's Cyberinfrastructure. University of Wisconsin Madison.

APPENDIX

A. Eucalyptus: Open Source Cloud Platform

1.1 Install Eucalyptus via tarball

The packages are available in a single tarball, wherein we also include copies of third-party CentOS packages that Eucalyptus depends on (Rampart, Axis2C, many Java libraries), at http://open.eucalyptus.com/downloads (look for a CentOS tarball of the right Eucalyptus version and architecture).

Untar the bundle in a temporary location:

```
tar zxvf eucalyptus-$VERSION-*.tar.gz
cd eucalyptus-$VERSION-*
```

In the examples below we use x86_64, which should be replaced with i386 or i586 on 32-bit architectures.

Install RPMs on the front end

First, on the front end, install third-party dependency RPMs:

```
cd eucalyptus-$VERSION*-rpm-deps-x86_64


rpm -Uvh aoetools-21-1.el4.x86_64.rpm \
        euca-axis2c-1.6.0-1.x86_64.rpm \
        euca-rampartc-1.3.0-1.x86_64.rpm \
        vblade-14-1mdv2008.1.x86_64.rpm \
        vtun-3.0.2-1.el5.rf.x86_64.rpm \
        lzo2-2.02-3.el5.rf.x86_64.rpm\
         perl-Crypt-OpenSSL-Random-0.04-1.el5.rf.x86_64.rpm\
        perl-Crypt-OpenSSL-RSA-0.25-1.el5.rf.x86_64.rpm\
        perl-Crypt-X509-0.32-1.el5.rf.noarch.rpm\
        python25-2.5.1-bashton1.x86_64.rpm\
        python25-devel-2.5.1-bashton1.x86_64.rpm\
        python25-libs-2.5.1-bashton1.x86_64.rpm
cd ..
```

then install the -cloud, -walrus, -cc and -sc RPMs:

```
rpm -Uvh eucalyptus-$VERSION-*.x86_64.rpm \
        eucalyptus-common-java-$VERSION-*.x86_64.rpm \
        eucalyptus-cloud-$VERSION-*.x86_64.rpm \
        eucalyptus-walrus-$VERSION-*.x86_64.rpm \
        eucalyptus-sc-$VERSION-*.x86_64.rpm \
        eucalyptus-cc-$VERSION-*.x86_64.rpm \
        eucalyptus-gl-$VERSION-*.x86_64.rpm
```

Install RPMs on the nodes

Next, on each node install the dependency packages:

```
cd eucalyptus-$VERSION*-rpm-deps-x86_64
rpm -Uvh aoetools-21-1.el4.x86_64.rpm \
        euca-axis2c-1.6.0-1.x86_64.rpm \
        euca-rampartc-1.3.0-1.x86_64.rpm\
        perl-Crypt-OpenSSL-Random-0.04-1.el5.rf.x86_64.rpm\
        perl-Crypt-OpenSSL-RSA-0.25-1.el5.rf.x86_64.rpm\
        perl-Crypt-X509-0.32-1.el5.rf.noarch.rpm\
        python25-2.5.1-bashton1.x86_64.rpm\
        python25-devel-2.5.1-bashton1.x86_64.rpm\
        python25-libs-2.5.1-bashton1.x86_64.rpm
cd ..
```

then install the node controller RPM with dependencies:

```
rpm -Uvh eucalyptus-$VERSION-*.x86_64.rpm \
        eucalyptus-gl-$VERSION-*.x86_64.rpm \
        eucalyptus-nc-$VERSION-*.x86_64.rpm
```

1.2 Uploading image

To enable a VM image as an executable entity, a user/admin must add a root disk image, a kernel/ramdisk pair (ramdisk may be optional) to Walrus and register the uploaded data with Eucalyptus. Each is added to Walrus and registered with Eucalyptus separately, using three EC2 commands. The following example uses the test image that we provide. Unpack it to any directory:

```
euca-bundle-image -i <kernel file> --kernel true

euca-upload-bundle -b <kernel bucket> -m /tmp/<kernel file>.manifest.xml

euca-register <kernel-bucket>/<kernel file>.manifest.xml
```

Next, add the root filesystem image to Walrus:

```
euca-bundle-image -i <vm image file>

euca-upload-bundle -b <image bucket> -m /tmp/<vm image file>.manifest.xml

euca-register <image bucket>/<vm image file>.manifest.xml
```

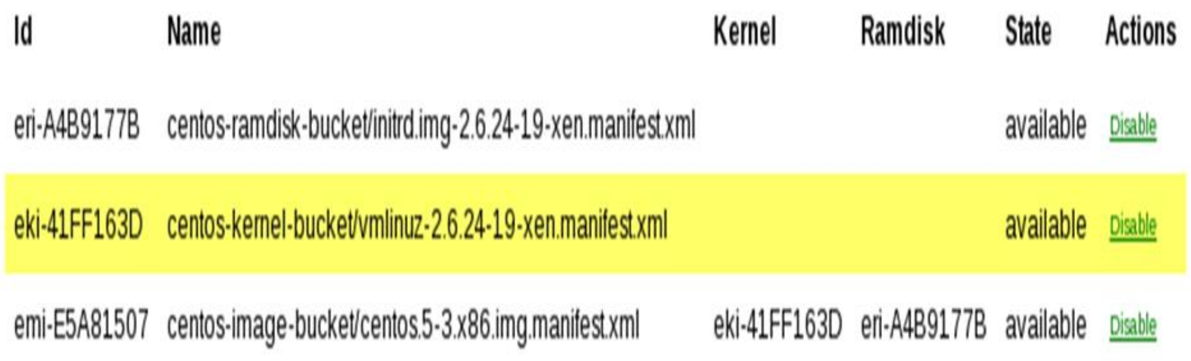
Associating kernels and ramdisks with instances

There are three ways that one can associate a kernel (and ramdisk) with a VM instance.

A user may associate a specific kernel/ramdisk identifier with an image at the 'euca-bundle-image' step
euca-bundle-image -i <vm image file> --kernel <eki-XXXXXXXX> --ramdisk <eri-XXXXXXXX>
A user may choose a specific kernel/ramdisk at instance run time as an option to 'euca-run-instances'
euca-run-instances --kernel <eki-XXXXXXXX> --ramdisk <eri-XXXXXXXX> <emi-

```
XXXXXXXX>
```

At this point, the system is ready to operate.



Id	Name	Kernel	Ramdisk	State	Actions
eri-A4B9177B	centos-ramdisk-bucket/initrd.img-2.6.24-19-xen.manifest.xml			available	Disable
eki-41FF163D	centos-kernel-bucket/vmlinuz-2.6.24-19-xen.manifest.xml			available	Disable
emi-E5A81507	centos-image-bucket/centos5-3.x86.img.manifest.xml	eki-41FF163D	eri-A4B9177B	available	Disable

B. Pegasus: Workflow Management System

Pegasus can also be installed via tarball option. The required package can be downloaded from http://pegasus.isi.edu/downloads

When the package is already downloaded. Untar the package via

```
# tar zxf pegasus-*.tar.gz
```

Include the Pegasus bin directory to PATH profile

```
# export PATH=/path/to/pegasus-4.0.0:$PATH
```

C. Condor: High Throughput Computing

Check if Condor is running

```
ps -ef | grep condor
```

Expected Result:

```
condor   29612     1  0 23:36 ?        00:00:01 /usr/sbin/condor_master -pidfile
/var/run/condor/master.pid
condor   29613 29612  0 23:36 ?        00:00:00 condor_collector -f
condor   29614 29612  0 23:36 ?        00:00:00 condor_negotiator -f
condor   29615 29612  0 23:36 ?        00:00:00 condor_schedd -f
condor   29616 29612  1 23:36 ?        00:00:05 condor_startd -f
root     29618 29615  0 23:36 ?        00:00:00 condor_procd -A
/var/run/condor/procd_pipe.SCHEDD -S 60 -C 102
```

The following table shows path layout compared to original Condor's binary package.

Orignal Location	**Current Location**
bin	/usr/bin
etc	/etc/condor/
etc/examples	/usr/share/doc/condor-{version}/etc/examples
examples	/usr/share/doc/condor-{version}/examples
include	/usr/include/condor

lib	/usr/lib/condor | /usr/lib64/condor
libexec	/usr/libexec/condor
local/condor_config.local	/etc/condor/
local/execute	/var/lib/condor/execute
local/spool	/var/lib/condor/spool
man	/usr/share/man
sbin	/usr/sbin
sql	/usr/share/condor/sql
src	/usr/src
INIT	/etc/init.d/condor
PID	/var/run/condor
LOGS	/var/log/condor
LOCK	/var/lock/condor

For the other installation instruction can be viewed via this website: http://research.cs.wisc.edu/condor/

This page intentionally left blank