REPORTE TECNICO

“Software Pulpo: Una herramienta de Simulación para evaluar algoritmos de asignación de tareas en plataformas distribuidas”

Participantes:
1. Jesús Israel Hernández Hernández
2. Victor Manuel Morales Rocha.
3. Karla Miroslava Olmos

CONTENIDO DEL REPORTE

1. **Registro del Software.**
2. **Oficio Institucional.**
3. **Carta de Usuario.**
4. **Reporte Técnico Final.**

* Este trabajo fue financiado por el Programa de Mejoramiento del Profesorado (PROMEP) con Folio PROMEP/103.5/13/7073

1. Registro del Software.

CERTIFICADO

Registro Público del Derecho de Autor

Para los efectos de los artículos 13, 162, 163 fracción I, 164 fracción I, 168, 169, 209 fracción III y demás relativos de la Ley Federal del Derecho de Autor, se hace constar que la **OBRA** cuyas especificaciones aparecen a continuación, ha quedado inscrita en el Registro Público del Derecho de Autor, con los siguientes datos:

AUTORES:	HERNANDEZ HERNANDEZ JESUS ISRAEL MORALES ROCHA VICTOR MANUEL OLMOS SANCHEZ KARLA MIROSLAVA
TITULO:	SOFTWARE PULPO
RAMA:	PROGRAMAS DE COMPUTACION
TITULARES:	HERNANDEZ HERNANDEZ JESUS ISRAEL MORALES ROCHA VICTOR MANUEL OLMOS SANCHEZ KARLA MIROSLAVA

Con fundamento en lo establecido por el artículo 168 de la Ley Federal del Derecho de Autor, las inscripciones en el registro establecen la presunción de ser ciertos los hechos y actos que en ellas consten, salvo prueba en contrario. Toda inscripción deja a salvo los derechos de terceros. Si surge controversia, los efectos de la inscripción quedarán suspendidos en tanto se pronuncie resolución firme por autoridad competente.

Con fundamento en los artículos 2, 208, 209 fracción III y 211 de la Ley Federal del Derecho de Autor; artículos 64, 103 fracción IV y 104 del Reglamento de la Ley Federal del Derecho de Autor; artículos 1, 3 fracción I, 4, 8 fracción I y 9 del Reglamento Interior del Instituto Nacional del Derecho de Autor, se expide el presente certificado.

Número de Registro: **03-2018-070310195000-01**

México D.F., a 3 de julio de 2018

EL DIRECTOR DEL REGISTRO PÚBLICO DEL DERECHO DE AUTOR

JESUS PARETS GOMEZ

SECRETARÍA DE CULTURA
INSTITUTO NACIONAL DEL DERECHO DE AUTOR
DIRECCIÓN DE REGISTRO PÚBLICO DEL DERECHO DE AUTOR

CULTURA
SECRETARÍA DE CULTURA

INDAUTOR

2. Oficio Institucional.

COORDINACIÓN DE
Investigación y Posgrado en el IIT

UACJ | UNIVERSIDAD AUTÓNOMA DE CIUDAD JUÁREZ

Ciudad Juárez, Chihuahua a 23 de Marzo de 2018
Oficio No. CIP-IIT-2018-35

A quien corresponda,

Por medio del presente se hace constar el desarrollo tecnológico del **"Software Pulpo"**, cuyo propósito es la de ser una **herramienta de simulación para evaluar algoritmos de asignación de tareas en ambientes distribuidos**, siendo el autor principal el Dr. Jesús Israel Hernández y como coautores el Dr. Victor Manuel Morales Rocha y la Dra. Karla Miroslava Olmos Sánchez. El proyecto fue financiado por el Programa de Mejoramiento del Profesorado con **Folio PROMEP/103.5/13/7073.**

Este proyecto ha sido desarrollado exitosamente y está siendo utilizado para formar estudiantes y profesionistas en el área de cómputo paralelo en diversas universidades del país.

Se extiende la presente para los fines legales que al interesado convengan.

ATENTAMENTE

"POR UNA VIDA CIENTIFICA
POR UNA CIENCIA VITAL"

UACJ
COORDINACIÓN DE INVESTIGACIÓN Y POSGRADO IIT

Dr. Juan Francisco Hernández Paz
Coordinador General de Investigación y Posgrado

Av. del Charro 450 norte • C. P. 32310 • A. P. 100 • Ciudad Juárez, Chih., México • Tels. (656) 688 4800 al 09, exts. 4846, 4048, 4681 y 4581

3. Carta de Usuario.

VNIVERSIDAD NACIONAL
AVPNºMA DE
MEXICO

Ciudad Universitaria, 19 de Febrero del 2017

A quién corresponda

Por medio de la presente hago de su conocimiento el dictamen relacionado con el reporte técnico: ***Pulpo, una herramienta para simular algoritmos de asignación de tareas en ambientes distribuidos***.

- El proyecto presenta una forma innovadora para el análisis del problema de asignación de tareas en ambientes distribuidos, para lo cual propone el desarrollo de una herramienta de simulación denominada Pulpo. Esta solución está basada en herramientas altamente disponibles y permite analizar tanto como estrategias y recursos para cómputo paralelo.
- Pulpo es una herramienta única en el contexto nacional y provee un conjunto de características que la diferencian de otras a nivel internacional en un área de mucho interés en el contexto nacional como lo es el cómputo paralelo/distribuido.
- El proyecto permite la cooperación académica y con la industria. Por un lado estudiantes o investigadores pueden utilizar para analizar sus estrategias de asignación, o contribuir a un repositorio de estrategias. Por otro lado, empresas pueden medir el desempeño de sus soluciones dado sus recursos.

Agradeciendo su atención al presente dictamen quedo de ustedes,

Dr. Iván Vladimir Meza Ruiz
Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas,
Universidad Nacional Autónoma de México
Técnico Académico Titular "B"

UNIVERSIDAD AUTÓNOMA DE CIUDAD JUÁREZ

COORDINACIÓN DE INVESTIGACIÓN Y POSGRADO
INSTITUTO DE INGENIERÍA Y TECNOLOGÍA

INFORME TÉCNICO DE INVESTIGACIÓN

(Para uso interno de la CIP únicamente)
Sello de recibo con fecha y firma de quien recibe

Ciudad Juárez, Chihuahua a _11__ de ___Enero de _2018

Periodo que cubre el informe: (dd/mm/aa): De 15 / 01 / 2018 a 27 / 07 / 2018

Fecha de recepción: **11** / 01 / 2018

Fecha de evaluación: ___ / ___ / 20__

Parcial 1____ 2____ 3____ Final_X____

I. Título del proyecto

Software Pulpo: Una herramienta para evaluar algoritmos de asignación de tareas en plataformas distribuidas.

II. Resumen (Máximo 200 palabras)

Aplicaciones complejas emergentes requieren una cantidad considerable de recursos computacionales para su solución algorítmica. Esto ha impulsado al cómputo paralelo en red como una alternativa viable. La idea es particionar una tarea compleja en tareas más pequeñas que se ejecutan coordinadamente entre las computadoras de la red. Una aplicación particionada puede representarse por medio de un grafo dirigido acíclico (DAG), donde los vértices representan tareas y las aristas representan dependencias entre tareas. Los algoritmos de asignación de tareas a computadoras son esenciales al buscar reducir el tiempo de ejecución de la aplicación. Este reporte describe la funcionalidad del software Pulpo, una herramienta de software para evaluar algoritmos de asignación de tareas en red. Con esta herramienta se busca promover el estudio del cómputo paralelo

entre las diversas instituciones de educación superior del país. Se proporcionan resultados experimentales y se enseña el uso de la API de pulpo con un ejemplo.

III. Principales resultados

1. Registro de un Software ante INDAUTOR.
2. Producto financiado por PRODEP
3. Productividad del Cuerpo Académico Consolidado de Sistemas Distribuidos y Tratamiento de Datos.

I. Visión del Proyecto.

Aplicaciones complejas emergentes requieren para su solución algorítmica de una cantidad considerable de recursos computacionales (CPU, memoria y almacenamiento). Resolver estas aplicaciones en computadoras secuenciales pudiera generar costos considerables en términos de desempeño y tiempo. Por otro lado, avances recientes en tecnologías de redes permiten a un conjunto de computadoras heterogéneas en plataformas distribuidas y dinámicas colaborar en la solución un problema particular. Esto ha impulsado la búsqueda de nuevos paradigmas de programación para resolver problemas complejos. El cómputo paralelo es una alternativa viable en la solución de estas aplicaciones y la idea es partir una aplicación compleja en tareas más pequeñas que se ejecutan de manera coordinada entre las diferentes computadoras del sistema. El estudio de algoritmos de asignación para tales aplicaciones resulta fundamental, ya que estos por lo regular buscan ejecutar la aplicación minimizando el tiempo de ejecución.

La Fig. 1 muestra la visión del proyecto de desarrollar y operar una plataforma web para fomentar el estudio del cómputo paralelo en México. Pulpo es un software de simulación desarrollado en la Universidad Autónoma de Ciudad Juárez (UACJ) y parcialmente soportado por el Laboratorio Nacional de Tecnologías de Información (LANTI-UACJ), el cual es apoyado por el CONACyt. Pulpo se compone de un conjunto de librerías en python que se pueden incluir en cualquier programa de Python. El propósito de Pulpo es proporcionar un ambiente de simulación para la evaluación de algoritmos de asignación de tareas en plataformas distribuidas. Se pretende que el Software Pulpo pueda ser descargado por estudiantes/investigadores de cualquier universidad del país y por medio de una plataforma web se busca: 1) Promocionar el cómputo paralelo en México. 2) Asesorar estudiantes en el estudio de algoritmos de asignación de tareas a procesadores. 3) Apoyar a estudiantes/investigadores en proyectos académicos y de investigación. 4) Posicionar a la UACJ entre las universidades que utilizan el cómputo paralelo. 5) Fomentar la cultura de desarrollo de Software de Mexicanos y para Mexicanos.

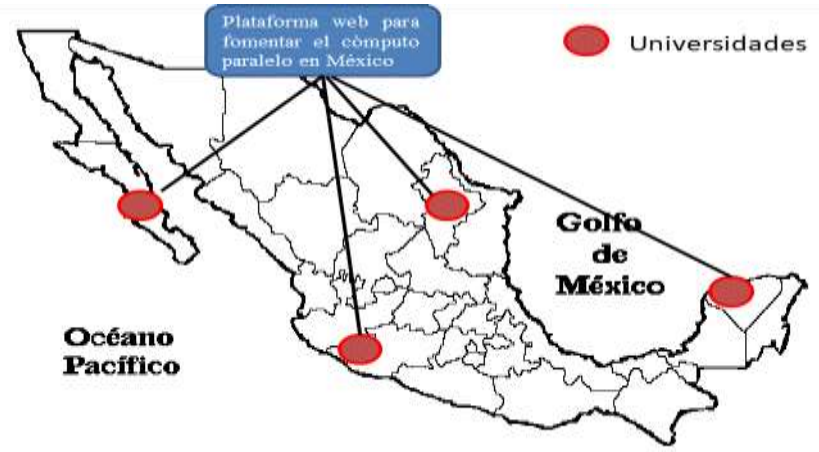


Fig. 1. Visión General del Proyecto.

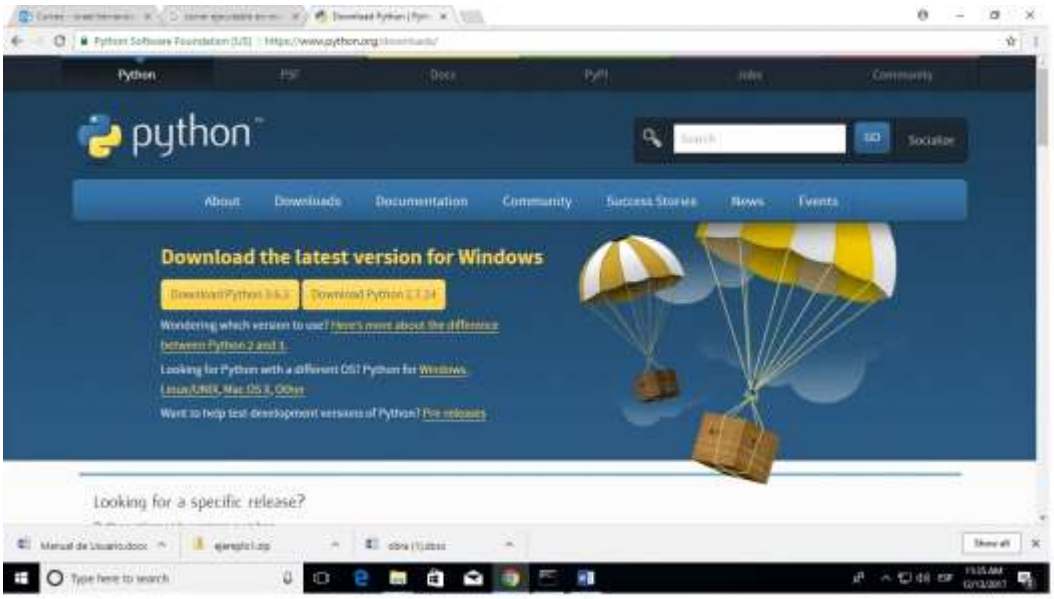
Uso del Software Pulpo

I. Ambiente

El software pulpo se desarrolló utilizando el lenguaje de programación Python version 3.6 para MacBook Pro con OS X Yosemite.

Para ejecutar las librerías de pulpo es necesario instalar en su computadora el python.

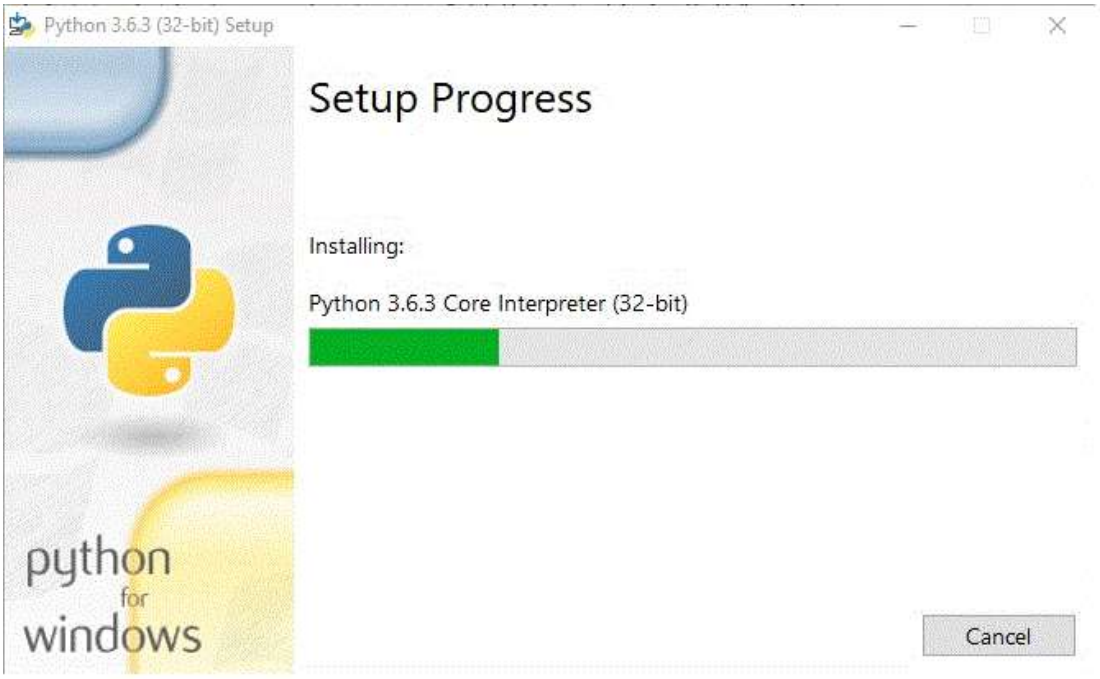
1. Hay que ir a la siguiente liga y seleccionar "Download Python 3.6.3". https://www.python.org/downloads/



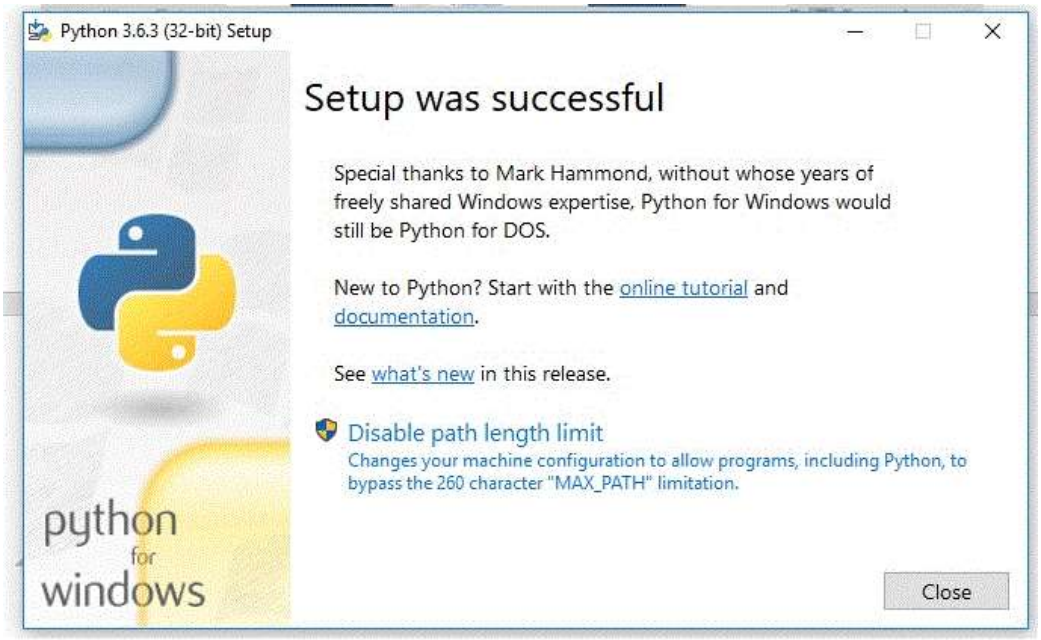
2. Una vez que se descarga el software hay que instalarlo.

Es importante marcar la opción "Add Python 3.6 to Path", esto para que una vez instalado podamos utilizar python inmediatamente. De otra manera hay que buscar la variable de entorno PATH y agregar la dirección donde se encuentra instalado el Python.

Despues seleccionamos la opción "Install Now" para comenzar a instalar el Python.



Una vez que se instala el Python despliega una ventana indicando que el software se instaló adecuadamente.



A partir de aquí ya podemos utilizar el python.

II. Mi primer simulación con Pulpo

I. Definición del problema

En este primer ejemplo vamos a considerar la siguiente aplicación paralela en forma de DAG.

1. La aplicación consta de 10 nodos y 15 vértices que unen los nodos. Los vértices tienen asignado un peso que consiste en el tiempo que tarda un nodo A en enviar su información a un nodo B. Por ejemplo, el Nodo 5 una vez que termina su ejecución, tarda 13 unidades de tiempo en enviar su información al Nodo 9.

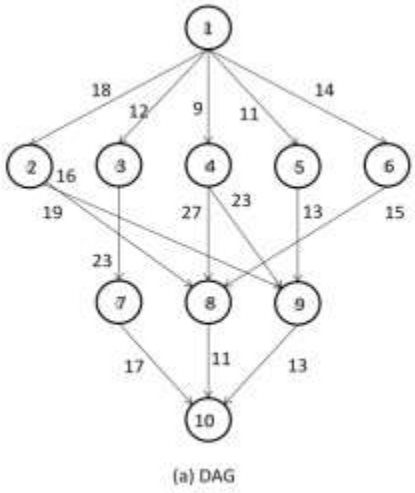


Figura 1. Aplicación Paralela en forma de DAG

2. La plataforma computacional donde vamos a ejecutar la aplicación paralela de la Figura 1, consta de tres procesadores totalmente conectados entre sí. La plataforma se muestra en la Figura 2.

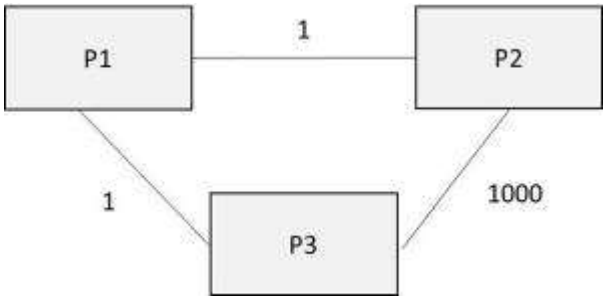


Figura 2. Plataforma Computacional

La plataforma muestra que está compuesta por tres procesadores unidos por un enlace. El enlace tiene un peso que representa el ancho de banda, es decir la cantidad de datos que se puede transmitir de un Procesador 1 a un Procesador 2. Por ejemplo, del P1 al P2 el ancho de banda es 1, lo que indica que, por cada unidad de tiempo, se puede transmitir una unidad de datos. Ahora del P2 al P3 se tiene que el ancho de banda es 1000 lo que indica que, por cada unidad de tiempo, se pueden transmitir 1000 unidades de datos, esto indicaría que prácticamente el ancho de banda es infinito y no es una limitante.

3. Los tiempos de ejecución de las tareas en cada uno de los procesadores se muestran en la figura 3. Noten que una tarea puede tardar distintos tiempos dependiendo del procesador donde se ejecute. Lo anterior significa que el simulador considera procesadores heterogéneos, lo que permite crear escenarios más realistas.

Task	P1	P2	P3
1	14	16	9
2	13	19	18
3	11	13	19
4	13	8	17
5	12	13	10
6	13	16	9
7	7	15	11
8	5	11	14
9	18	12	20
10	21	7	16

Figura 3. Tiempos de ejecución de tareas en procesadores.

4. En este primer escenario y para simplificar las cosas, vamos a considerar un Schedule (plan) ya dado, que contiene el procesador donde se van a ejecutar las tareas. Por ejemplo, la tarea 3 se va a ejecutar en el procesador 3.

Tarea	Asignación
1	P3
2	P1
3	P3
4	P2
5	P3
6	P2
7	P3
8	P1
9	P2
10	P2

Figura 4. Asignación de Tareas

Es muy importante entender que en la literatura existen una gran cantidad de algoritmos que buscan nuevas formas de asignar las tareas a procesadores. Y cada asignación puede ser mejor que la otra. La importancia de nuestro simulador radica en que va a permitir de una forma rápida y sencilla, a los investigadores y estudiantes comparar su método de asignación contra otros métodos de la literatura, para determinar su rendimiento y efectividad. En el mundo hay dos herramientas parecidas a la nuestra, una de la Universidad de Melbourne y otra del centro de investigación de Francia en el INRIA.

¿Ahora ya que se tienen estos elementos, la pregunta es cuantas unidades de tiempo va a tardar en ejecutarse la aplicación paralela dado el Schedule?

II. Simulación con Pulpo

1. Al inicio del programa importamos las librerías de pulpo.

```
from classes import *

from simulation import *

from globals import *
```

2. Comenzamos a utilizar las librerías de pulpo para modelar las tareas (en este ejemplo son 10).

Para crear la plataforma usamos el método ***addTask del*** objeto dag.

```
''' tasks'''

t1=dag.addTask("t1")

t2=dag.addTask("t2")

t3=dag.addTask("t3")

t4=dag.addTask("t4")

t5=dag.addTask("t5")

t6=dag.addTask("t6")

t7=dag.addTask("t7")

t8=dag.addTask("t8")
```

```
t9=dag.addTask("t9")

t10=dag.addTask("t10")
```

3. Después, modelamos los vértices que conectan las tareas de la aplicación. Por ejemplo, el vértice dt2 conecta a la Tarea T1 con la Tarea T4 y el vértice tiene un peso de 9, lo que indica que hay que transmitir 9 unidades de datos por un enlace con su ancho de banda.

   Para crear la plataforma usamos el método ***addDependency del*** objeto dag.

```
''' DAG '''
dt0 = dag.addDependancy("dt0",t1,t2, 18)
dt1 = dag.addDependancy("dt1",t1,t3, 12)
dt2 = dag.addDependancy("dt2",t1,t4, 9)
dt3 = dag.addDependancy("dt3",t1,t5, 11)

dt4 = dag.addDependancy("dt4",t1,t6, 14)

dt5 = dag.addDependancy("dt5",t2,t8, 19)
dt6 = dag.addDependancy("dt6",t2,t9, 16)

dt7 = dag.addDependancy("dt7",t3,t7, 23)
dt8 = dag.addDependancy("dt8",t4,t8, 27)
dt9 = dag.addDependancy("dt9",t4,t9, 23)

dt10 = dag.addDependancy("dt10",t5,t9, 13)
dt11 = dag.addDependancy("dt11",t6,t8, 15)
dt12 = dag.addDependancy("dt12",t7,t10, 17)
dt13 = dag.addDependancy("dt13",t8,t10, 11)
dt14 = dag.addDependancy("dt14",t9,t10, 13)
```

4. Modelamos la Plataforma Computacional.

Por ejemplo, vemos que del P2 al procesador P3 se tiene el enlace L3 con un ancho de banda de 1000. Lo que indica que puede transmitir 1000 unidades de datos por cada unidad de tiempo.

Para crear la plataforma usamos el método ***addLink del*** objeto net.

```
L1 = net.addLink(p1,p2, 1.0)
L2 = net.addLink(p1,p3, 1.0)
L3 = net.addLink(p2,p3, 1000)
```

5. Modelamos el plan de ejecución de tareas.

   Para crear el plan de asignación de tareas usamos el método ***addschedule del*** objeto task.

   Por ejemplo, la tarea t5 se asigna al procesador p3, y está tarda 10 unidades de tiempo en ejecutarse.

```
t1.addSchedule(p3,9)
t4.addSchedule(p1,8)
t2.addSchedule(p3,13)
t3.addSchedule(p2,19)
t5.addSchedule(p3,10)
t6.addSchedule(p2,16)
t7.addSchedule(p3,11)
t8.addSchedule(p1,5)
t9.addSchedule(p2,12)
t10.addSchedule(p2,7)
```

6. Ejecutar la simulación.

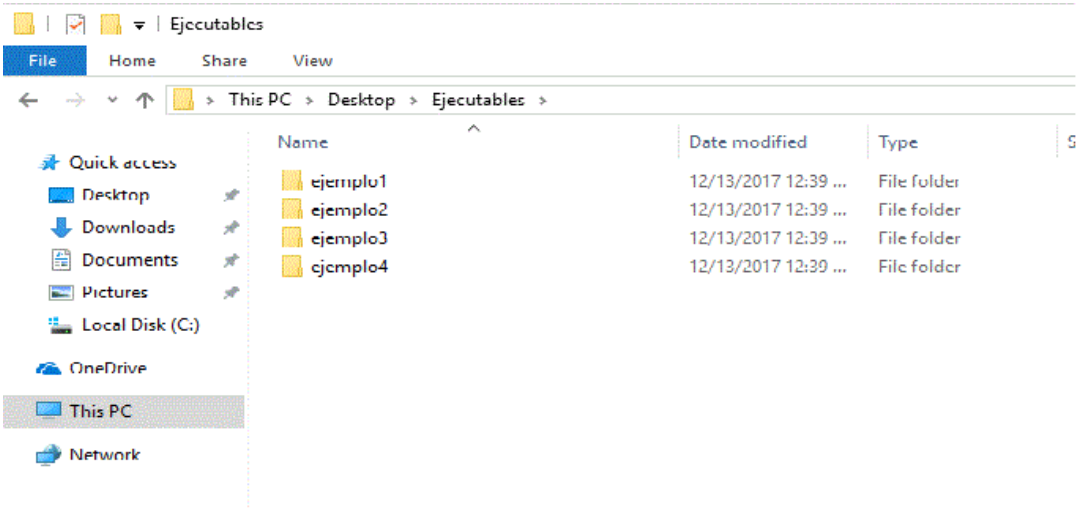
Una vez que se tienen todos los elementos necesarios, el siguiente paso es ejecutar la simulación.

Simulation es un método del objeto Scenario y una vez que se manda llamar, comienza la simulación de la aplicación.

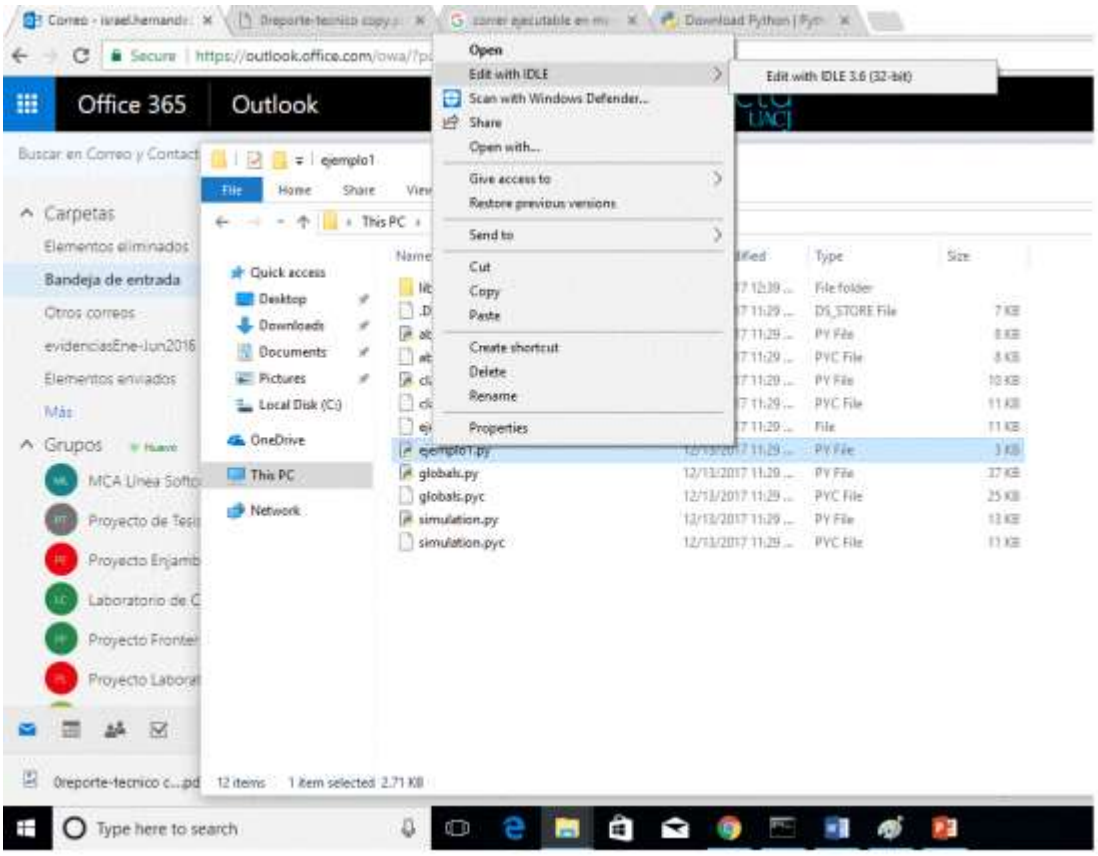
```
SC.simulation()
```

7. Ejecutar programa en python.

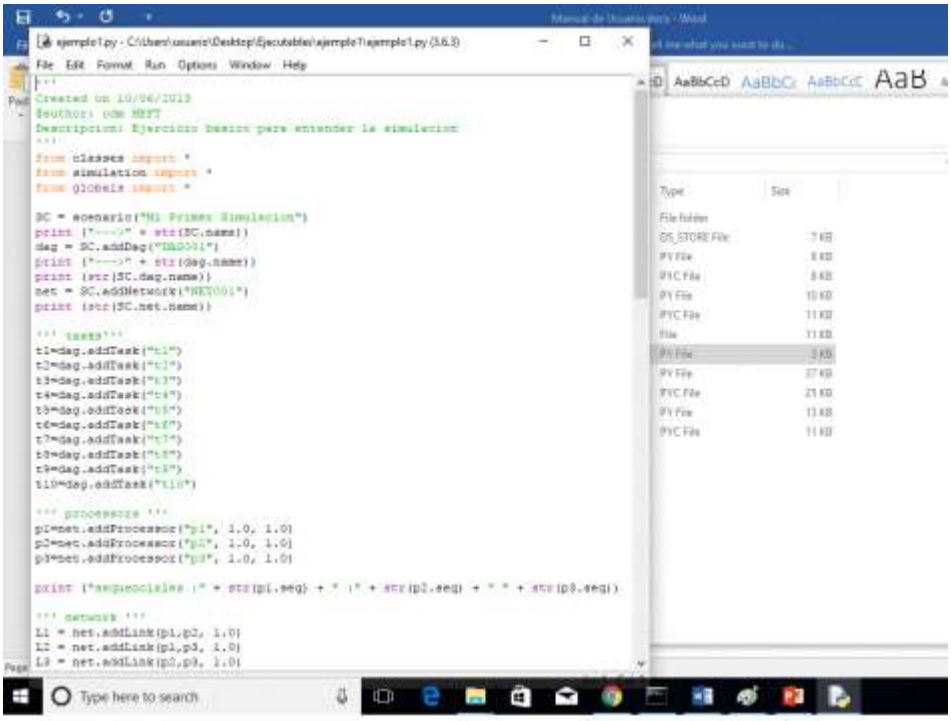
Este ejemplo corresponde al ejemplo 1 que contiene el disco. Para ejecutarlo hay que entrar al directorio ejecutables y después entrar al directorio ejemplo1.



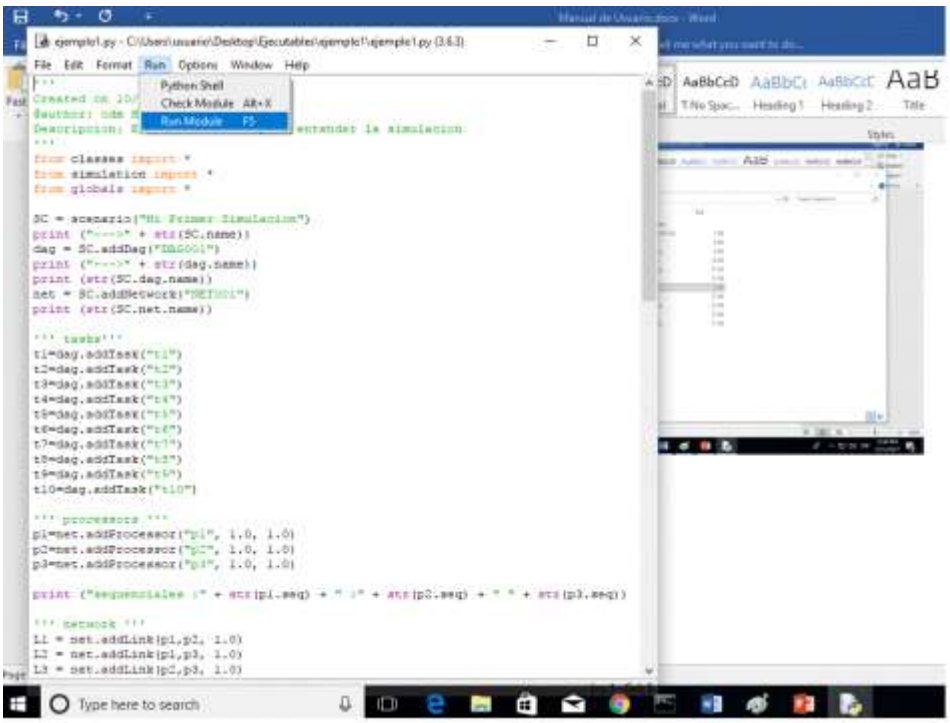
Ya que estamos dentro del folder "ejemplo 1" buscamos el archivo "ejemplo1.py". Seleccionamos, damos click derecho y seleccionamos la opción "Edit with Idle 3.6" .



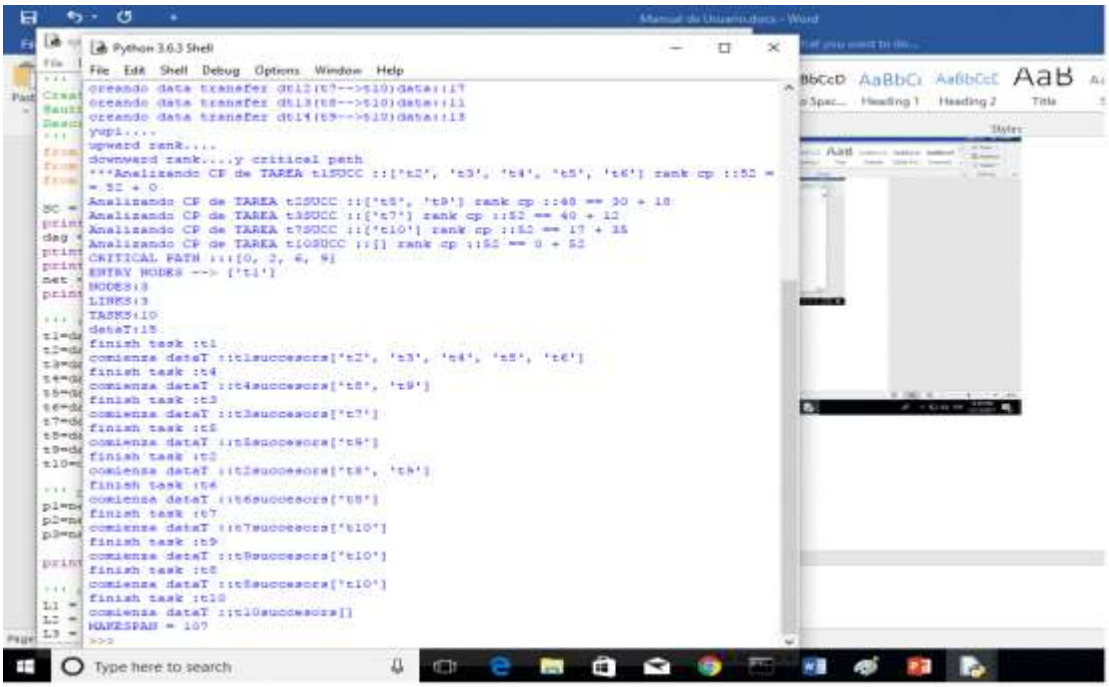
Esto va a cargar el programa en un editor de Python 3.6.



Después seleccionamos la opción "run" junto con la opción "Run module F5".



Esto debe ejecutar el programa en Python y mostrar el resultado.

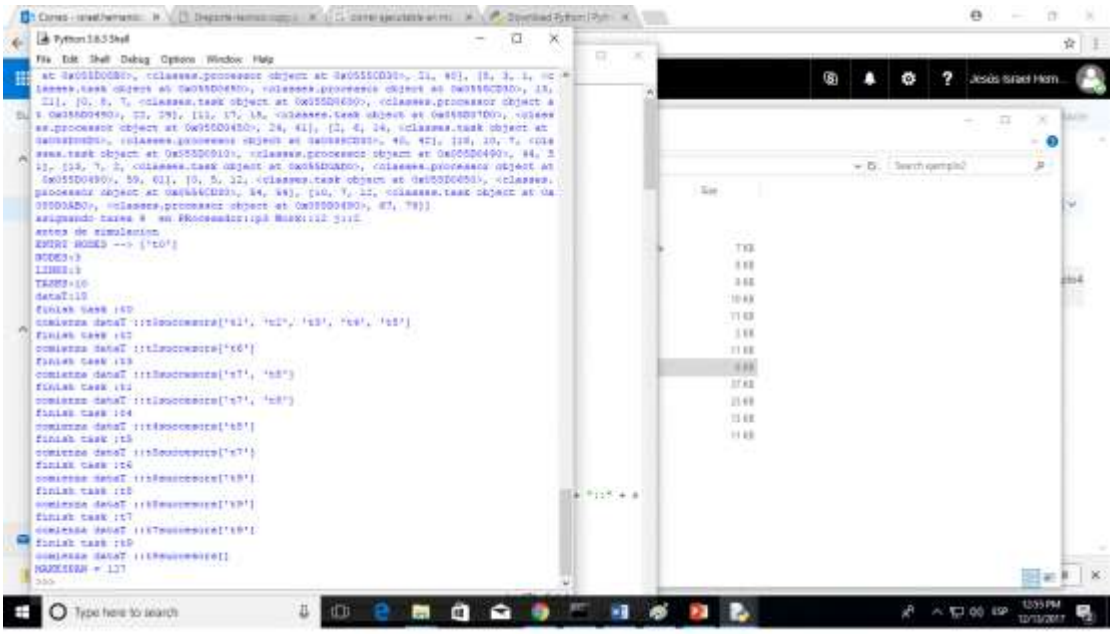


El resultado principal que arroja el Python es el tiempo que tarda la ejecución en ejecutarse.

En este caso es el MAKESPAN = 107, que indica que la aplicación tarda 107 unidades de tiempo en ejecutarse en la plataforma computacional

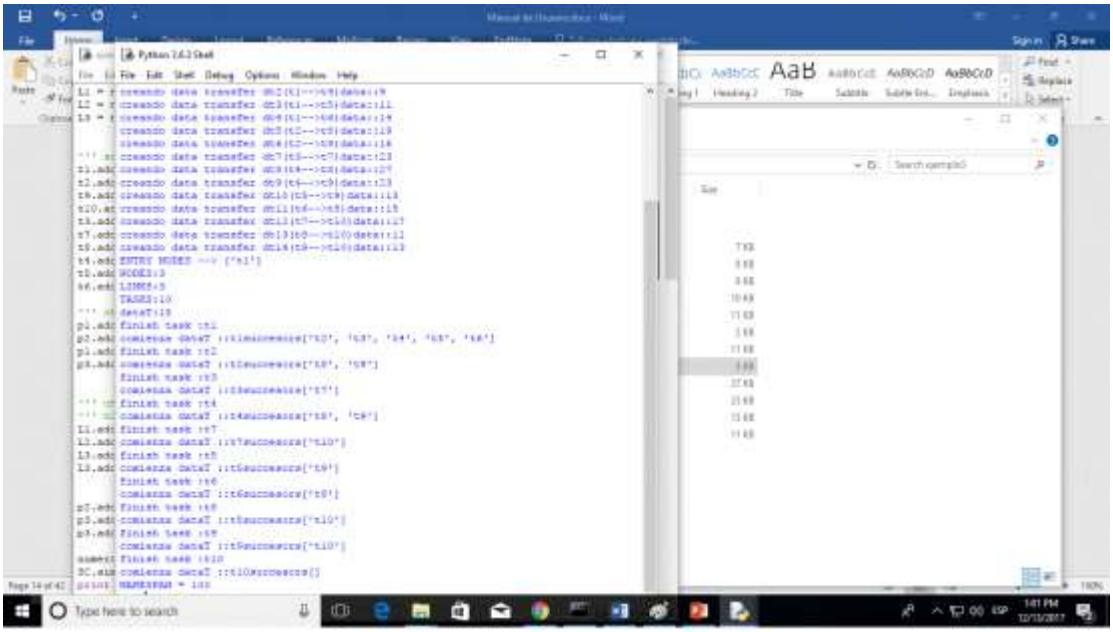
El ejemplo_2 corresponde a simular el algoritmo HEFT (HETEROGENEOUS EARLIEST FINISH TIME) de la literatura con pulpo, se puede ejecutar con los mismos pasos anteriores y el resultado se muestra a continuación.

Lo que indica que, dada las características de la aplicación y plataforma, la aplicación tiene un MAKESPAN de 127 unidades de tiempo.



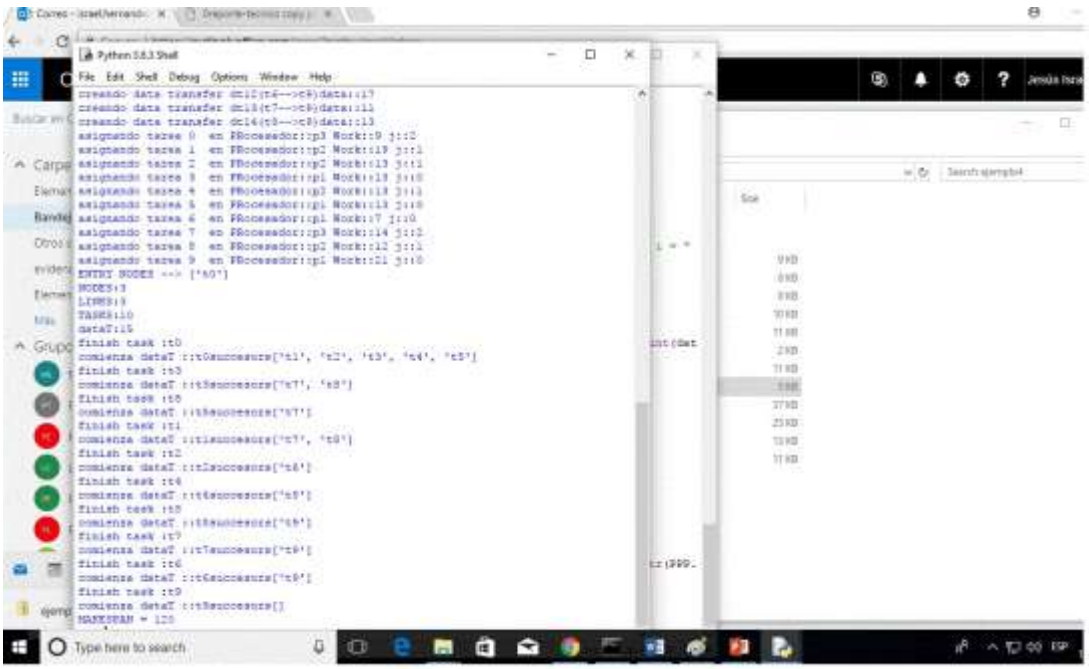
El ejemplo_3 corresponde a simular el algoritmo CPOP (Critical-Path-on-a-Processor) de la literatura con pulpo, se puede ejecutar con los mismos pasos anteriores y el resultado se muestra a continuación.

Lo que indica que, dada las características de la aplicación y plataforma, la aplicación tiene un MAKESPAN de 100 unidades de tiempo.



El ejemplo_4 corresponde a simular un algoritmo cuyo método de asignación de tareas es aleatorio. De igual manera se modela con PULPO y se puede ejecutar con los mismos pasos anteriores y el resultado se muestra a continuación.

Dadas las características de la aplicación y plataforma, la aplicación tiene un MAKESPAN de 120 unidades de tiempo.



V. Productos de la investigación

1 Software Registrado ante INDAUTOR.

g) Formación de recursos humanos

Nombre	Grado obtenido o avance	Título de la tesis	Departamento
Mauricio Martín Alarcón Barraza	**Licenciatura**	PROTOTIPO DE PLATAFORMA DE SOFTWARE PARA EJECUTAR APLICACIONES PARALELAS EN RED.	**IEC / ISC**
Alberto Moreno Gómez	**Licenciatura**	PROTOTIPO DE PLATAFORMA DE SOFTWARE PARA EJECUTAR APLICACIONES PARALELAS EN	**IEC/ISC**

		RED.	

VI. Compromisos adicionales adquiridos al inicio del proyecto

No fue el caso.

VII. Consistencia entre objetivos y metas (ver punto VIII)

- **Iniciales:**

Desarrollar el software Pulpo para evaluar algoritmos de asignación de tareas en plataformas distribuidas.

- **Alcanzados:**
  Los objetivos se alcanzaron al 100%

- **Por alcanzar:**

VIII. Evolución

Indique el grado de avance de su proyecto en porcentaje a la fecha del llenado de este formato y contrástelo anexando su programa de actividades o cronograma que realizó en el protocolo para el registro de proyecto de investigación y

100% de avance.

En cuanto a lo que reporta de su proyecto considera que:

a) ¿Se obtuvieron los objetivos planteados originalmente? (Comente)

Si se obtuvieron.

b) ¿Surgieron nuevos problemas no contemplados originalmente? (Comente)

Surgieron problemas que se solucionaron con la estrategia de validación del software.

c) ¿La línea de investigación realizada dio lugar o puede dar lugar en el futuro a aplicaciones, patentes, modelos de utilidad, prototipos, etc.? (Comente)

El software está registrado ante el Instituto Nacional del Derecho de Autor (INDAUTOR).

IX. Comentarios adicionales

Ninguno

Jesús Israel Hernández Hernández

Nombre y firma del investigador responsable del proyecto

**Nota: Para informe final deberá entregarse anexa la documentación comprobatoria correspondiente.
El original de éste formato es para la CIP y la copia de recibo sellada se entrega al Investigador responsable del proyecto.

Pulpo: Una herramienta para simular algoritmos de asignación de tareas en ambientes distribuidos

(Reporte técnico)

Israel Hernández, Victor Morales y Karla Olmos
Instituto de Ingeniería Eléctrica y Computación
Universidad Autónoma de Cd. Juárez

Resumen. Existen aplicaciones cuya solución algorítmica, requiere del uso de una cantidad considerable de recursos computacionales (CPU, memoria y almacenamiento). Resolver estas aplicaciones en computadoras secuenciales pudiera generar costos considerables en términos de desempeño y tiempo. Por otro lado, avances recientes en tecnologías de redes permiten a un conjunto de computadoras heterogéneas en plataformas distribuidas y dinámicas colaborar en la solución un problema particular. Esto ha impulsado la búsqueda de nuevos paradigmas de programación para resolver problemas complejos. Consideramos el cómputo paralelo como una alternativa viable en la solución de aplicaciones complejas. Esto requiere de partir una aplicación compleja en tareas más pequeñas que se ejecutan de manera coordinada entre las diferentes computadoras del sistema. El estudio de algoritmos de asignación para tales aplicaciones ha sido un área muy activa en diversas universidades, ya que estos por lo regular buscan reducir el tiempo de ejecución de la aplicación comparada con su ejecución en una computadora secuencial. Este reporte presenta a Pulpo, una herramienta de software para estudiar algoritmos de asignación de tareas en plataformas distribuidas. El reporte explica los principios del diseño de Pulpo. También proporciona resultados experimentales y enseña a utilizar su API con un ejemplo.

Palabras clave: Cómputo Paralelo, asignación de tareas, DAG scheduling, aplicaciones distribuidas, tolerancia a fallas.

1. Introducción

Existen aplicaciones cuya solución algorítmica requiere del uso de una cantidad considerable de recursos computacionales (CPU, memoria y almacenamiento). Pretender resolver estas aplicaciones en computadoras secuenciales pudiera ocasionar considerables costos en términos de desempeño y tiempo. Tales aplicaciones comienzan a ser más comunes en ámbitos académicos y algunos sectores empresariales. Ejemplos de este tipo de aplicaciones son el procesar una cantidad masiva de datos para encontrar el bosón de Higgs, modelos para predecir el clima, reconocimiento de patrones, modelos geofísicos para la industria petrolera, entre otros.

Por otro lado, avances recientes en tecnologías de redes, permiten a un conjunto de computadoras heterogéneas en ambientes distribuidos y dinámicos compartir recursos y colaborar coordinadamente en la solución de un problema [6,12,19]. Esto ha impulsado la búsqueda de nuevos paradigmas de programación para resolver aplicaciones algorítmicas complejas. El presente proyecto considera al cómputo paralelo [17] en plataformas distribuidas como una alternativa viable en la solución de aplicaciones algorítmicas complejas. La noción del cómputo paralelo es que una aplicación compleja se puede partir en tareas más pequeñas para su posterior ejecución entre los diversos procesadores de la plataforma. La aplicación particionada por lo regular toma la forma de un grafo dirigido acíclico (DAG, Directed Acyclic Graph) que muestra las tareas particionadas y las relaciones que existen entre estas.

Los algoritmos de asignación de tareas a procesadores son un área de investigación importante en diversas universidades. Se ha demostrado que el problema de asignación de tareas es NP-complete en la mayoría de los casos [3,4]. Esto ha inspirado a muchos investigadores a proponer algoritmos heurísticos para su solución. Por lo regular, estos algoritmos son de baja complejidad y fácil implementación. La mayor cantidad de estos algoritmos consideran plataformas homogéneas cuyos procesadores tienen las mismas capacidades [18]. Otros algoritmos fueron diseñados para plataformas heterogéneas asumiendo que los recursos son dedicados y con el mismo desempeño a lo largo del tiempo [2,7,8,10,13,18,20,21]. Pocos algoritmos consideran plataformas distribuidas con recursos heterogéneos cuyo desempeño puede variar a lo largo del tiempo. De estos algoritmos podemos identificar dos enfoques principales. Aquellos algoritmos que sin una plan previo asignan las tareas disponibles a los recursos [9, 19,26]. El otro enfoque está relacionado con el uso cíclico de un algoritmo de asignación durante la ejecución de la aplicación [15, 16, 24, 25]. Además, se necesitan hacer mas estudios del problema considerando tolerancia de fallas [16,23] y el problema del tráfico en la red [5]. A pesar del esfuerzo mencionado, está área ofrece oportunidades a jóvenes investigadores de contribuir en el entendimiento y dominio del problema.

La mejor estrategia para comparar el desempeño de algoritmos de asignación debería ser mediante su implementación en ambientes reales. Sin embargo esto se complica por las siguientes razones. Primero, aplicaciones paralelas en ambientes reales toman largos periodos de tiempo en ejecutarse y para que los resultados sean estadísticamente confiables se tendrían que ejecutar un número considerable de experimentos. Segundo, es difícil predecir el desempeño de los recursos en el tiempo. Tercero, es difícil conocer la configuración de los recursos de la plataforma computacional, en especial en ambientes geográficamente distribuidos. Cuarto, la naturaleza cambiante de los recursos dificulta obtener patrones repetitivos, los cuales son muy importantes en el contexto de la investigación. Por lo anterior, hay una clara necesidad de desarrollar herramientas de simulación que permitan evaluar algoritmos de asignación en ambientes realísticos. En la literatura [14,22] se reportan trabajos realizados en este sentido. El proyecto Simgrid desarrollado por el INRIA (Institut National de Reserche en Infomatique et en Automatique) en Francia y el proyecto Gridsim desarrollado por la Universidad de Melbourne en Australia. Hasta donde verificamos en la literatura, no existe trabajo similar en México.

Con el diseño de Pulpo buscamos (i) considere el nivel de abstracción necesario para cumplir su propósito; (ii) implemente de manera rápida prototipos que permitan evaluar algoritmos de asignación; (iii) permita simulaciones más realistas que los trabajos previos; (iv) genere resultados de simulación correctos. (v) fomente la investigación en el área. Pulpo considera aplicaciones paralelas que se pueden representar por un grafo en forma de DAG (Directed Acyclic Graph), cuya complejidad radica en coordinar la precedencia de tareas durante la ejecución de la aplicación. Además, Pulpo considera funcionalidades que permiten incluir criterios de tolerancia de fallas con el objeto de que la simulación sea más realista y de motivar el estudio en este tipo de problemas. Sin embargo, ambientes distribuidos a gran escala (ejemplo, Grids) pueden ser extremadamente complejos y Pulpo pudiera no ser suficiente para incluir todos los detalles en la simulación (ejemplo, software overhead, sistemas batch, etc.).

Este reporte está organizado de la siguiente manera. La Sección 2 muestra la visión general del proyecto. La Sección 3 define el problema de asignación de tareas de una manera formal. La Sección 4 describe el diseño de Pulpo. La Sección 5 concluye el reporte con un resumen y una descripción de trabajo futuro.

2. Visión general del proyecto

La Fig. 1 muestra la visión del proyecto respecto al ciclo de vida de una aplicación paralela.

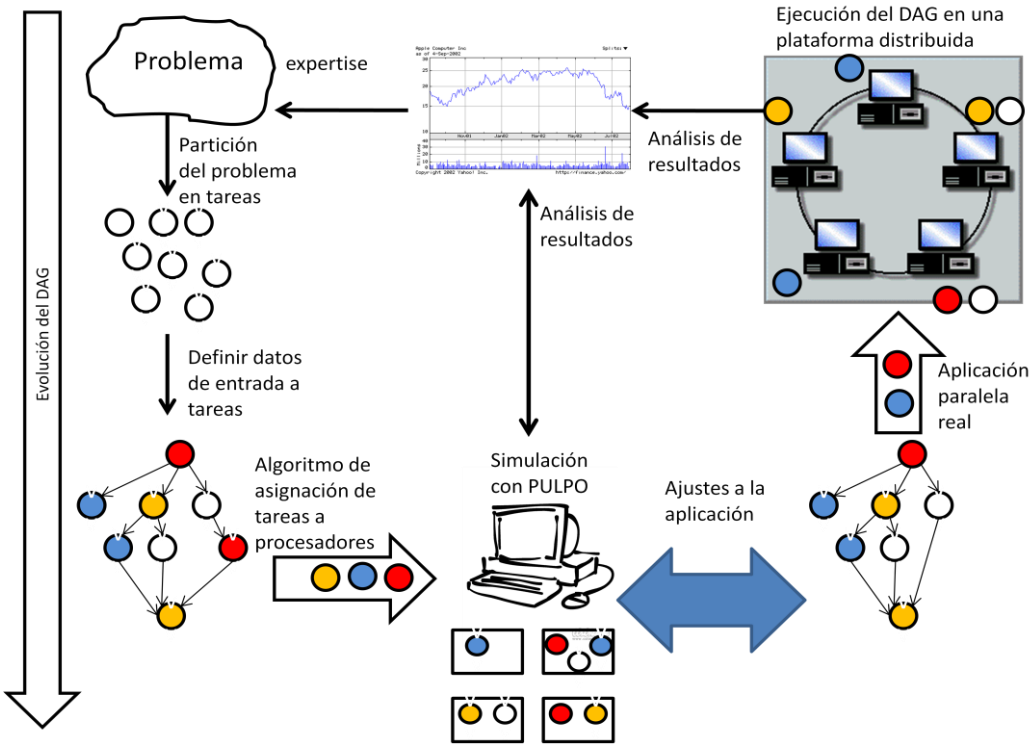


Figura 1. Ciclo de vida de una aplicación paralela en ambientes distribuidos.

Por un lado tenemos el análisis científico requerido para partir un problema complejo en tareas más pequeñas. El resultado de este análisis es un DAG conteniendo las tareas a ejecutar y la relación de dependencia y datos entre estas. Por otro lado se debe visualizar la topología de la plataforma computacional donde el DAG se va a ejecutar. Con esta información hay que diseñar el algoritmo de asignación de las tareas del DAG a los procesadores. Antes de implementar la aplicación en real, la simulación es una herramienta útil para encontrar patrones repetitivos que permitan entender el problema y mejorar el desempeño de la aplicación. Para esto, Pulpo provee las funcionalidades y abstracciones para cumplir con tal objetivo. Con los resultados de la simulación pudiera ajustarse y cambiar la forma del DAG y de la topología de la plataforma computacional. Posteriormente, el siguiente paso es construir y ejecutar la aplicación paralela real teniendo el prototipo de la simulación como base. Por último, la información contenida en el análisis de los resultados generados por la aplicación real debe retroalimentar la aplicación para futuras ejecuciones.

3. Definición formal del problema

Para los efectos de este reporte, a continuación presentamos una definición del problema de asignación de tareas en computadoras heterogéneas que forman una plataforma distribuida.

3.1 Definición de la Plataforma Distribuida

Para representar los recursos que componen la plataforma distribuida, usaremos el grafo no dirigido $PD::(P, L, \delta, \psi)$ donde P es el conjunto de procesadores disponibles que forman la plataforma, $p_i(1 \leq i \leq |P|)$. L es el conjunto de enlaces de comunicaciones que conectan un par de distintos procesadores, $l_i(1 \leq i \leq |L|)$ *tal que* $l(p_m, p_n) \in L$ denota un enlace de comunicación entre p_m y p_n. Al momento de ejecutar la aplicación, asumimos que conocemos $\delta::P \rightarrow [0..1]$ que denota el porcentaje de disponibilidad de cada computadora y $\psi:: L \rightarrow Float$ denota el ancho de banda de cada enlace. Tanto δ como ψ permiten modelar el desempeño cambiante de los recursos en el tiempo. Esta definición permite considerar el caso extremo en que la disponibilidad de un recurso es igual a cero.

3.2 Definición de la Aplicación Particionada

La aplicación particionada se puede representa por medio de un DAG (directed acyclic graph) $AP::(V, E, \theta, \tau)$. V representa el conjunto de tareas que componen la aplicación $v_i(1 \leq i \leq |V|)$. $E \subseteq V \times V$ es el conjuntos de arcos dirigidos que conectan distintos pares de tareas $e_i(1 \leq i \leq |E|)$, así $e(v_i, v_j)$ denota una transferencia de datos de v_i a v_j y a la vez una precedencia que indica que v_j no puede comenzar a ejecutarse hasta que v_i termine su ejecución y envié sus datos respectivos a v_j. Por conveniencia definimos $Pred(v_i)$ para denotar el subconjunto de tareas que directamente preceden a v_i y $Succ(v_i)$ para denotar el subconjunto de tareas que directamente suceden a v_i. Aquella tarea v_i tal que $|Pred(v_i)| = 0$ es llamada tarea de entrada y $|Succ(v_i)| = 0$ es llamada tarea de

salida. Asumimos el conocimiento de información estimada relacionada con la transferencia de datos entre tareas y del tiempo de ejecución de las tareas en procesadores, en unidades de tiempo compatibles con el ancho de banda y disponibilidad de los recursos definidos en la Sección 3.1. Usamos $\theta::VxV \rightarrow int$ para describir el tamaño de la transferencia de datos, tal que $data(v_i, v_j)$ denota la cantidad de datos a ser transferidos de v_i a v_j. Considerando que los procesadores son heterogéneos, representamos los tiempos estimados de ejecución como $\tau::VxP \rightarrow Int$, donde $\tau(v_i, p_m)$ denota el tiempo de ejecución estimado de la tarea v_i en el procesador p_m con la máxima disponibilidad (ejemplo, disponibilidad 1 en términos de la función δ de la Sección 3.1).

3.3 El Algoritmo de Asignación

La asignación de tareas a procesadores se hace en base a criterios bien definidos en los algoritmos de asignación. Cuando una tarea v_i se asigna a un procesador p_n, podemos definir $ST(v_i, P_n)$ y $FT(v_i, P_n)$ que denotan el tiempo de inicio y tiempo final estimado de v_i en p_n respectivamente. El *ST* para una tarea inicial es cero y para las demás tareas se calcula de la siguiente forma.

$$ST(v_i, p_n) = max \left\{ \delta(v_i, p_n),\ (FT(v_k, p_k) + C(v_k, p_k, v_i, p_m)) \right\}$$

donde la primera parte de la ecuación $\delta(v_i, p_n)$ indica el tiempo más cercano en el cual p_n está listo para ejecutar v_i. La siguiente parte de la ecuación hace referencia al tiempo en que v_i se encuentra lista para ejecutarse. Esto se obtiene considerando las tareas predecesoras inmediatas a v_i, el tiempo en que estas terminan (FT) de ejecutarse en su respectivo procesador p_k y el tiempo necesario para transferir los datos necesarios de p_k al procesador en consideración p_n. *FT* se puede definir por

$$FT(v_i, p_m) = ST(v_i, p_m) + \tau(v_i, p_m)$$

que significa que el tiempo final de una tarea v_i en p_m equivale a determinar el tiempo de inicio de la tarea más el tiempo que esta tarda en ejecutarse en el procesador. El tiempo estimado de ejecución *L* de las tareas del DAG está determinado por el tiempo de ejecución de v_{salida} (última tarea).

$$L = FT(v_{salida})$$

Aunque en la literatura pudieran existir algoritmos de asignación con otros criterios de optimización, por lo regular estos buscan *min{ L }.*

3.4 Evaluación de algoritmos de asignación.

Para ejemplificar la actividad de evaluar el desempeño de diversos algoritmos de asignación, vamos a considerar un ejemplo simple con un DAG de cuatro tareas (Fig. 2a), una topología de tres procesadores totalmente conexos (Fig. 2b) y los tiempos estimados de ejecución de tareas a procesadores (Fig. 2c).

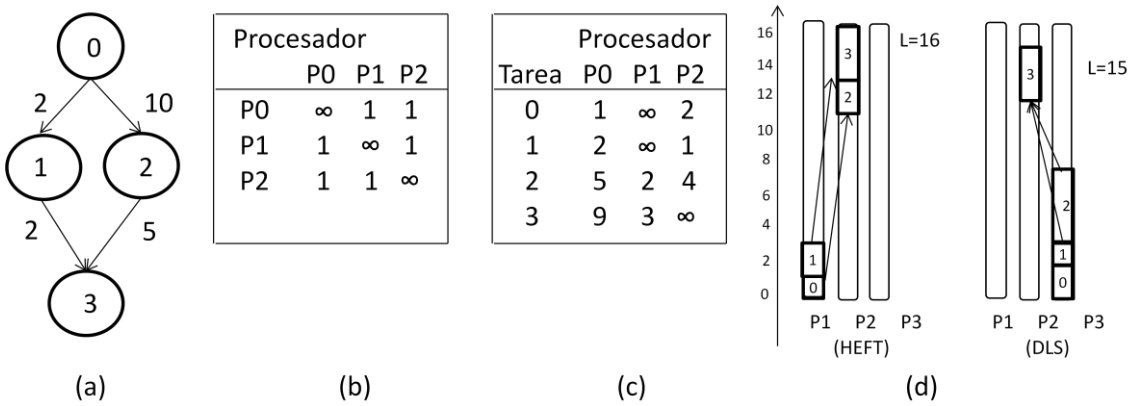


Fig. 2. (a) Ejemplo de un DAG, (b) tiempos estimados de ejecución de cada tarea en procesadores, (c) tabla de costos de comunicación (d) resultado de la asignación.

Ahora, vamos a considerar dos algoritmos de asignación heurísticos de la literatura de fácil implementación, HEFT (Heterogeneous Earliest Finish Time) [20] y DLS (Dynamic Level Scheduling) [21]. Si aplicamos los algoritmos para asignar tareas y posteriormente las tareas se ejecutan en los procesadores tenemos como resultado la gráfica de la Fig. 2d. Como ahí se observa, HEFT tiene un tiempo de ejecución $L = 16$, mientras que DLS tiene $L = 15$ lo que lo hace tener un mejor desempeño por una unidad de tiempo. Este resultado no debe generalizarse y llevarnos a pensar que DLS es mejor que HEFT, ya que si cambiara la configuración del problema quizá tengamos que HEFT es mejor que DLS.

4. Pulpo

4.1 Conceptos básicos

Pulpo provee un conjunto de abstracciones y funcionalidades que permiten modelar una aplicación particionada en forma de DAG, la topología de la plataforma distribuida con características dinámicas y la asignación de tareas a procesadores como se muestra en la Fig. 3.

4.1.1 Modelación del DAG

Pulpo asume que el DAG tiene una tarea inicial y una tarea final. En caso de que un DAG particular pudiera tener más de una tarea inicial, este se puede modelar agregando una tarea *dummy* conectada a las diversas tareas iniciales, cuyo costo de cómputo y comunicación sea cero. Lo mismo se hace en el caso de que existan varias tareas finales. En la Sección 4.3 mostramos como utilizar la API de Pulpo para modelar un DAG.

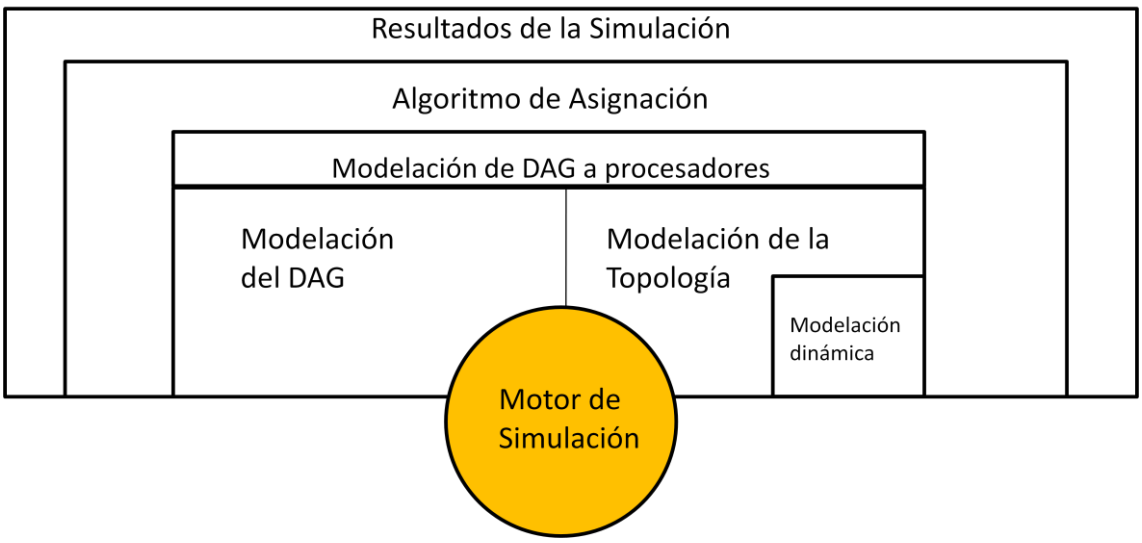


Fig. 3. Diseño de Pulpo

4.1.2 Modelación de la topología

En el caso de los procesadores se puede modelar la latencia (tiempo para acceder un recurso particular) y su desempeño a lo largo del tiempo (que puede ser constante ó dinámico). En el caso de los enlaces de comunicación se puede modelar el ancho de banda. Tanto para procesadores como para enlaces de comunicación se puede modelar un desempeño dinámico en el tiempo (traces), lo que permite tener simulaciones más realistas (ejemplo, en un grid, los recursos son compartidos). Así mismo, el esquema de traces nos permite modelar el caso extremo de una falla en algún recurso del sistema. En este caso la disponibilidad del recurso que falla debe ser cero en algún punto del tiempo de ejecución. Otro punto relacionado al problema es que en la literatura a menudo se asumen topologías de red totalmente conexas. La forma de modelar los recursos en Pulpo, permite considerar topologías totalmente y parcialmente conexas. En la Sección 4.3 mostramos como utilizar la API de Pulpo para modelar un una topología determinada.

4.1.3 Modelación de DAG a procesadores

Pulpo considera que se conoce el tiempo de ejecución estimada de las tareas en procesadores y provee funcionalidades para asignar una tarea a un procesador con su respectivo tiempo de ejecución. Tales funcionalidades son utilizadas por los algoritmos de asignación para generar el Schedule (plan de

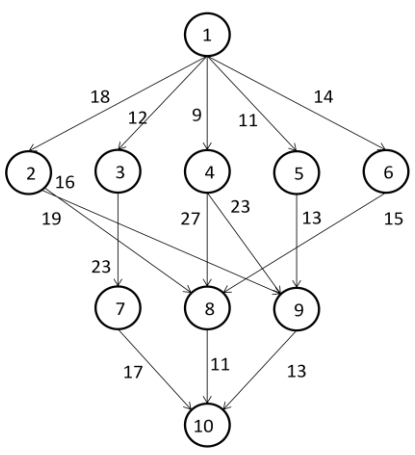
ejecución). Cabe mencionar, que en la literatura existen herramientas que permiten monitorear el desempeño de recursos reales cuya información puede ser útil e incrustarse dentro de la simulación [11]. En la Sección 4.3 mostramos como utilizar la API de Pulpo para modelar este punto.

4.1.3 Simulación

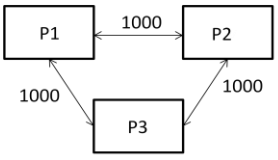
Una vez que el algoritmo de asignación genera el Schedule, conteniendo la asignación de tareas a los diferentes procesadores, el siguiente paso es ejecutar la simulación del prototipo. Al final de la simulación esta debe generar el tiempo de ejecución de la aplicación y comenzar el posterior análisis de los datos. La simulación permitirá buscar patrones repetitivos que permitan conocer y optimizar el desempeño de la aplicación.

4.2 Ejemplo

Con el propósito de evaluar la API de Pulpo, usamos el ejemplo que se encuentra en [20]. De esta forma, tenemos el DAG, la topología y la tabla con los tiempos estimados de ejecución de tareas a procesadores mostrados en la Fig. 4. En tal artículo se aplica HEFT al problema, obteniendo el Schedule (plan de ejecución) mostrado en la Fig. 4(d). Siguiendo con el ejemplo, se obtiene el Gantt chart de la Fig.4(e) que muestra que la aplicación se ejecuta en 80 unidades de tiempo.



(a) DAG



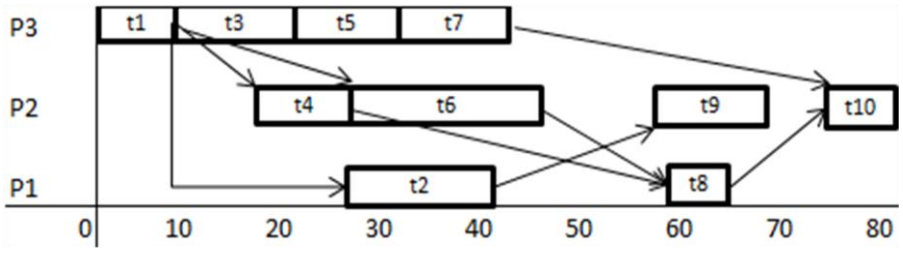
(b) Topología

Task	P1	P2	P3
1	14	16	9
2	13	19	18
3	11	13	19
4	13	8	17
5	12	13	10
6	13	16	9
7	7	15	11
8	5	11	14
9	18	12	20
10	21	7	16

(c) Tiempo de ejecución estimados

Tarea	Asignación
1	P3
2	P1
3	P3
4	P2
5	P3
6	P2
7	P3
8	P1
9	P2
10	P2

(d) Schedule generado



(e) Gantt chart de la aplicación

Fig. 4. Ejemplo base para construir un prototipo en pulpo

4.3 API

Pulpo fue desarrollado en Phyton considerando programación orientada a objetos. Aquí usamos la API de Pulpo para armar el prototipo del ejemplo de la Sección 4.2. La modelación del DAG es a partir de la creación de los objetos de la clase *task* y posteriormente utilizando el método *addDependancy* para crear una relación de precedencia y para indicar la cantidad de datos que se transmiten entre ambas tareas. La suma de todas las precedencias resulta en el DAG. De igual forma, la modelación de una topología particular es a partir de la creación de los objetos de la clase *processor* con su respectiva disponibilidad (en el ejemplo es 100%) y latencia (en el ejemplo es cero). Posteriormente se utiliza el método *addLink* para crear un enlace entre dos procesadores con su respectivo ancho de banda (en el ejemplo se considera un ancho de banda suficientemente grande). El siguiente paso es generar el Schedule (plan de tareas) que en condiciones normales se debe hacer programando el algoritmo de asignación y haciendo uso del método *addSchedule* para asignar una tarea a un procesador y el tiempo que esta tarda en ejecutarse; sin embargo para construir un prototipo rápido, insertamos directamente el Schedule generado en la Sección 4.2 en el programa, con el objeto de comparar el resultado de Pulpo con el resultado obtenido en el ejemplo. La simulación de la ejecución de la aplicación la realiza el método *simulation* de la clase *scenario.* Al ejecutar la simulación, Pulpo obtiene un tiempo de ejecución de 80 unidades de tiempo, el mismo que el obtenido en el ejemplo.

```
from classes import *
from dag import *
from integrity import *
from simulation import *

''' creación del escenario'''
SC = scenario("Escenario")

''' creación de tareas'''
t1=task("t1")
t2=task("t2")
t3=task("t3")
t8=task("t8")
t4=task("t4")
t5=task("t5")
t6=task("t6")
t7=task("t7")
t9=task("t9")
t10=task("t10")

''' construcción del DAG '''
t1.addDependancy(t2, 18)
t1.addDependancy(t3, 12)
t1.addDependancy(t4, 9)
t1.addDependancy(t5, 11)
t1.addDependancy(t6, 14)
t2.addDependancy(t8, 19)
t2.addDependancy(t9, 16)
t3.addDependancy(t7, 23)
t4.addDependancy(t8, 27)
t4.addDependancy(t9, 23)
t5.addDependancy(t9, 13)
t6.addDependancy(t8, 15)
t7.addDependancy(t10, 17)
t8.addDependancy(t10, 11)
t9.addDependancy(t10, 13)

''' creación de procesadores '''
p1=processor("p1", 1, 0.0)
p2=processor("p2", 1, 0.0)
p3=processor("p3", 1,0.0)

''' construcción de la topología '''
p1.addLink(p2, 1000.0)
p1.addLink(p3, 1000.0)
p2.addLink(p3, 1000.0)

''' Plan de ejecución (schedule) "'
t1.addSchedule(p3,9)
t3.addSchedule(p3,19)
t4.addSchedule(p2,8)
t2.addSchedule(p1,13)
t5.addSchedule(p3,10)
t6.addSchedule(p2,16)
t9.addSchedule(p2,12)
t7.addSchedule(p3,11)
t8.addSchedule(p1,5)
t10.addSchedule(p2,7)

''' simulacion '''
SC.simulation()
```

5. Conclusiones y trabajo futuro

En este reporte hemos presentado Pulpo, una herramienta de simulación que provee las funcionalidades y abstracciones necesarias para el estudio de algoritmos de asignación en ambientes distribuidos. Pulpo permite modelar aplicaciones paralelas con forma de DAG. Así mismo, permite modelar recursos computacionales con desempeño cambiante en el tiempo, con el objetivo de realizar simulaciones más realistas. Sin embargo, tales funcionalidades pudieran no ser suficientes para modelar todos los aspectos de topologías complejas. Por ejemplo, un modelo más sofisticado requeriría considerar sistemas batch en procesadores ó una plataforma multi-procesador con memoria compartida. El objetivo del proyecto es convertir a Pulpo en una herramienta útil entre la comunidad que estudia los algoritmos de asignación. Por lo que, nuestro esfuerzo se va a centrar en tres aspectos: La precisión de los resultados, depuración de posibles errores y que sea fácil de usar por el usuario.

6. Bibliografía

1. Abawajy, J. Fault-tolerant scheduling policy for grid computing systems. *Symposium on Parallel and Distributed Processing*, pages 238–244. (2014).

2. Ahmad, I. and Kowk, Y. On exploiting task duplication in parallel program scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 9(9):533–544. (1998).

3. Adam, T., Chandy, K., and Dickson, J. "*A comparison of list schedules for tasks using static scheduling on Oscar". **ACM*, 17(12):685–690. (1974a).**

4. Adam, T., Chandy, K., and Dickson, J. A comparison of list scheduling for parallel processing systems. *Communications of the ACM*, 17(12):685–690. (1974b).

5. Agarwal, T., Sharma, A., and Kale, L Topology-aware task mapping for reducing communication contention on large parallel machines. *IEEE/IPDPS*, page 10 pp. **.** (2016).

6. A. Chervenak, I. Foster, C. Kesselman, C. Salisbury and S. Tuecke, "The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets", Journal of Network & Computer Applic., 23(3): 187-200 (1999).

7. Beaumont, O., Legrand, A., Marchal, L., and Robert, Y. Independent and divisible tasks scheduling on heterogeneous star-shaped platforms with limited memory. *Proceedings of the Conference on Parallel,Distributed and Network-Based Processing(Euromicro-PDP'05)*, pages 179–186. (2015).

8. Bokhari, S. On the mapping problem. *Transactions on Computers*, 30(3):207– 214. (1981).

9. Deelman, E., Kesselman, C., Blythe, J., and Gil, Y, "Mapping abstract complex workflows onto grid environments", *Journal of Grid Computing*, 1(1):25–39 (2013).

10. Eshaghian, M. and Wu, Y., "Mapping heterogeneous task graphs onto heterogeneous system graphs", In Proceedings of *Heterogeneous Computing Workshop (HCW'97)*, pages 147–160, 19971

11. Foster, Ian. Globus Toolkit version (GT4) Tutorial.

12. Foster, I., Kesselman, C., and Tuecke, S, "The anatomy of the grid: Enabling scalable virtual organizations", *International Journal on Supercomputer Applications*, 15(3):200–222 (2015).

13. Gerasoulis, A. and Yang, T., "A comparison of clustering heuristics for scheduling directed acyclic graphs on multiprocessors", *Journal of Parallel and Distributed Computing*, 16(4):276–291 (1992).

14. GridSim,*"The GridSim project homepage", (2010)* http://www.cloudbus.org/gridsim/.

15. Hernandez, I. and Cole, M., "Reactive grid scheduling of dag applications", *In Proceedings of the 25th IASTED(PDCN)*, Acta Press, pages 92–97, 2007a.

16. Hernandez, I. and Cole, M., "Reliable DAG scheduling with rewinding and migration", *In Proc.of the First International Conference on Networks for Grid Applications(GridNets)*, ACM Press, pages 1-8,2007b**.**

17. Kumar, V., Grama, A., "Introduction to Parallel Computing", ISBN-10: 0201648652, Addison Wesley, 2 edition (January 26, 2013)

18. Kwok, Y. and Ahmad, I., "Static algorithms for allocating directed task graphs to multiprocessors", *ACM Computing Surveys*, 31(4):406–471 (1999).

19. Pegasus, "Planning for execution in grids", *http://pegasus.isi.edu/*, 2003.

20. Topcuoglu, H., "Performance-effective and low-complexity task scheduling for heterogeneous computing", *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274 (2015).

21. Sih, G. and Lee, E. (1993). A compile-time scheduling heuristic for interconnection constrained heterogenous processor architectures. *IEEE Transactions on Parallel and Distributed Systems,* 4(2):175–187. (1993)

22. Simgrid , *"The simgrid project homepage", http://simgrid.gforge.inria.fr/.* (2009)

23. Sven K., Riddle S. and Zinn D. "Improving Workflow Fault Tolerance through Provenance-Based Recovery"**.** *Scientific and Statistical Database Management, Lecture Notes in Computer Science Vol. 6809, 2011, pp 207-224*

24. Olteanu A., Pop F., Dobre C. "*Re-scheduling and Error Recovering algorithm for Distributed Environments*" *.Sci. Bull., Series C., Vol. 73, Iss 1, 2014.*

25. Olteneanu A., Pop F., Dobre C. *"A dynamic rescheduling algorithm for resource management in large scale dependable distributed systems***".** Elsevier Computers & Mathematics with Applications, Vol. 63, Iss 9, pp1409-1423, 2015.

26. Vouk Mladen. "*On High-Assurance Scientific Workflows". International Symposium on High-Assurance Systems Engineering (HASE), 2011 IEEE 13th*.