Pulpo y la necesidad de un ambiente colaborativo para el estudio del cómputo paralelo en México

Jesús Israel Hernández[1], Youness El Hamzaoui[1], Victor Morales[1]

Resumen

Aplicaciones complejas emergentes se caracterizan por requerir 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 conectadas en red, colaborar en la solución de un problema particular. Esto ha promovido el cómputo paralelo en red como una alternativa viable en la solución de aplicaciones complejas. La idea es particionar una tarea compleja en tareas más pequeñas que pueden ejecutarse simultáneamente entre las diferentes computadoras de la red. Los algoritmos de asignación de tareas a computadoras se vuelven fundamentales al buscar reducir el tiempo de ejecución de la aplicación particionada. Este artículo busca fomentar el estudio del cómputo paralelo entre las instituciones de educación superior en México. Pulpo es una herramienta de simulación creada para evaluar algoritmos de asignación de tareas en plataformas distribuidas. Se pretende que pulpo sea una herramienta útil a la comunidad académica interesada en el área y a su vez permita crear un ambiente colaborativo en el estudio del cómputo paralelo. Proporcionamos resultados experimentales y enseñamos a utilizar las librerías de pulpo con un ejemplo.

Palabras clave: Cómputo paralelo, planificación de tareas, algoritmos heurísticos.

Introducción

Aplicaciones complejas emergentes requieren para su solución algorítmica una cantidad considerable de recursos computacionales (CPU, memoria y almacenamiento). Pretender resolver estas aplicaciones en computadoras secuenciales pudiera ocasionar costos considerables 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 conectadas en red, compartir recursos y colaborar coordinadamente en la solución de un problema (Chervenak et al., 2000; Foster et al., 2001; Mell & Grance, 2011). Esto ha impulsado la búsqueda de nuevos paradigmas de programación para solucionar aplicaciones complejas. El presente proyecto considera el cómputo paralelo (Pacheco, 2011; Razdan, 2014) en red como una alternativa viable en la solución de aplicaciones algorítmicas complejas. La noción del cómputo paralelo es particionar una tarea compleja en tareas más pequeñas

[1] Departamento de Ingeniería Eléctrica y Computación, del Instituto de Ingeniería y Tecnología. Universidad Autónoma de Ciudad Juárez.

tal que estas puedan asignarse y ejecutarse de manera coordinada entre las diversas computadoras que componen la red.

La aplicación particionada se puede representar por medio de diversos grafos dependiendo de la relación que exista entre las diversas tareas que conforman la aplicación. Este proyecto considera aplicaciones particionadas que se pueden representar por medio de un grafo dirigido acíclico (DAG, por sus siglas en ingles), donde las aristas representan las tareas particionadas y los vértices representan una relación precedencia entre tareas. La complejidad de estas aplicaciones radica en considerar las restricciones de precedencia al momento de asignar y ejecutar las tareas que forman el DAG a las computadoras de la red.

Se ha demostrado que el problema de asignar tareas de un DAG a computadoras es NP-completo en la mayoría de los casos (Adam et al., 1974). Esto ha inspirado a muchos investigadores a proponer algoritmos heurísticos de baja complejidad para su solución. En la literatura se observa que conforme avanza la tecnología de redes se hace necesario desarrollar nuevos tipos de algoritmos heurísticos para explotar las características emergentes de las redes. De esta forma, una gran cantidad de algoritmos en la literatura consideran las computadoras de la red como dedicadas a la aplicación y con el mismo desempeño a lo largo del tiempo (ej. clusters) (Gerasoulis & Yang, 1992; Eshaghian & Wu, 1997; Kwok & Ahmad, 1999; Topcuoglu, 2002; Beaumont et al., 2005). Algunos algoritmos recientes consideran redes con computadoras heterogéneas no dedicadas, distribuidas geográficamente y con desempeño variante a lo largo del tiempo (ej. grids, nubes, etc.) (Hernandez & Cole, 2007a; Hernandez & Cole, 2007b; Eun-Kyu et al., 2011; Olteanu et al., 2011; Vouk & Mouallem, 2011; Olteanu et al., 2012). Estrategias de asignación de tareas en este tipo de plataformas se vuelven fundamentales, ya que las redes actuales siguen está tendencia. Además, se necesitan desarrollar estrategias que permitan la tolerancia de fallas (Hernandez & Cole, 2007b; Sven et al., 2011) y optimizar el problema del tráfico de datos en la red (Agarwal et al., 2006). A pesar del esfuerzo mencionado, la naturaleza del problema ofrece oportunidades a la comunidad universitaria de contribuir en el entendimiento y dominio del problema.

La mejor estrategia para evaluar el desempeño de los algoritmos de asignación debería ser mediante su implementación en ambientes reales. Sin embargo esto se complica por las siguientes razones:

1) El desarrollo de una aplicación paralela real no es sencillo y se necesita invertir un tiempo considerable en la implementación.

2) 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.

3) Es difícil predecir el desempeño de los recursos en el tiempo.

4) Es difícil conocer la configuración de las computadoras de la red, en especial en ambientes donde los recursos no son dedicados.

5) 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 plataformas distribuidas de manera realista. En GridSim (2010) y Simgrid (2009) los autores reportan trabajos similares 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. Pulpo contiene funcionalidades distintivas enfocadas en modelar plataformas computacionales con tendencias actuales. Tales funcionalidades permiten modelar la naturaleza dinámica de los recursos de red a lo largo del tiempo. Un caso particular es la posibilidad de modelar una falla controlada en algún recurso computacional de la red. Cabe mencionar, que modelar en su totalidad plataformas computacionales distribuidas recientes (ej., grids, clouds) puede ser extremadamente complejo y pulpo pudiera no ser suficiente para abstraer todos los detalles en la simulación (ej., sistemas por lotes, desperdicio de ancho de banda, etc.). Hasta el momento de la escritura del presente artículo, no existen trabajos similares en universidades de México.

Con el diseño de pulpo se pretende que un estudiante/investigador en el área:

(i) Tenga una herramienta con un nivel de abstracción adecuado para realizar su trabajo;

(ii) Implemente de manera rápida sus algoritmos de asignación;

(iii) Realice simulaciones más realistas que otros trabajos previos;

(iv) Pueda realizar pruebas de rendimiento que permitan evaluar su algoritmo con respecto a otros algoritmos;

(v) Genere resultados de simulación confiables.

Este artículo 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 presenta los resultados obtenidos. La sección 5 describe la API de pulpo y la sección 6 concluye el artículo con las conclusiones y trabajo futuro.

Visión general del proyecto

La literatura muestra la importancia que tiene actualmente el cómputo paralelo en la solución de problemas complejos. Es necesario que en México busquemos formas de avanzar en el estudio, comprensión y dominio de esta área del conocimiento. Por tal motivo nos enfocamos en el desarrollo tecnológico de pulpo y métodos de enseñanza de apoyo que nos permita impulsar el uso del cómputo paralelo en diversos sectores del país. Otros puntos relacionados con la visión del proyecto se mencionan a continuación.

1.- Parte de nuestra motivación y esfuerzo se centran en que un estudiante de cualquier universidad del país, pueda tener acceso a pulpo, métodos de enseñanza y recibir asesoría en español para implementar sus proyectos.

2.- Posicionar a la UACJ como una universidad líder en el estudio y dominio del cómputo paralelo, fortaleciendo los

programas de ingeniería en sistemas computacionales, ingeniería en software y la maestría en cómputo aplicado.

3.- Buscamos escribir un libro teórico-práctico que contenga métodos de enseñanza apoyados con el uso de pulpo, orientados a formación de recursos humanos.

4.- Promover los beneficios de nuestro proyecto en la industria, que nos permita obtener financiamiento para nuevos proyectos de tecnología.

5.- Buscamos sumar el esfuerzo y talento de investigadores de otras universidades en este proyecto. Esto permitirá crear sinergias para fortalecer y consolidar los objetivos del proyecto.

Más allá de los beneficios tecnológicos que el proyecto puede aportar, también nos esforzamos por complementar la visión de las entidades encargadas de definir las políticas de tecnología del país. En este sentido expresamos los siguientes puntos que consideramos parte de la visión del proyecto.

1.- La mayor parte de nuestros investigadores se concentra principalmente en escribir artículos en revistas indizadas, porque así lo dictan las entidades encargadas de definir las políticas de tecnología del país. En muchos casos estos artículos solo fortalecen el currículum del investigador y no tienen un impacto significativo en el desarrollo tecnológico del país. Este proyecto busca mostrar los beneficios que pueden tener los desarrollos tecnológicos propios en la formación de recursos humanos en áreas prioritarias del país.

2.- Un porcentaje alto de estudiantes no domina el inglés como segundo idioma. Esto puede marginar a estudiantes talentosos, con la capacidad de hacer aportaciones significativas en áreas prioritarias. Necesitamos hacer esfuerzos en crear desarrollos tecnológicos propios y ponerlos al alcance de los estudiantes de cualquier universidad. Esto permitirá motivar, descubrir, formar, orientar e integrar estudiantes que contribuyan con el desarrollo tecnológico del país.

La Fig. 1 sintetiza la visión del proyecto, la cual describimos a continuación. El ciclo de vida de una aplicación paralela inicia por un análisis metodológico para definir el dominio de la tarea compleja a resolver. Con este análisis se procede a particionar la tarea compleja en sub-tareas más pequeñas, cada una con su propio dominio. Después hay que definir las dependencias y transferencias de datos entre las tareas particionadas, dando como resultado la forma que tendrá la aplicación. Este proyecto considera aplicaciones en forma de un grafo acíclico dirigido (DAG por sus siglas en ingles). Algoritmos heurísticos de asignación de tareas a procesadores se vuelven esenciales en la solución de la aplicación (Kwok & Ahmad, 1999), ya que buscan optimizar el tiempo de ejecución de la aplicación. La complejidad del DAG radica en la ejecución coordinada de tareas respetando precedencias de tareas.

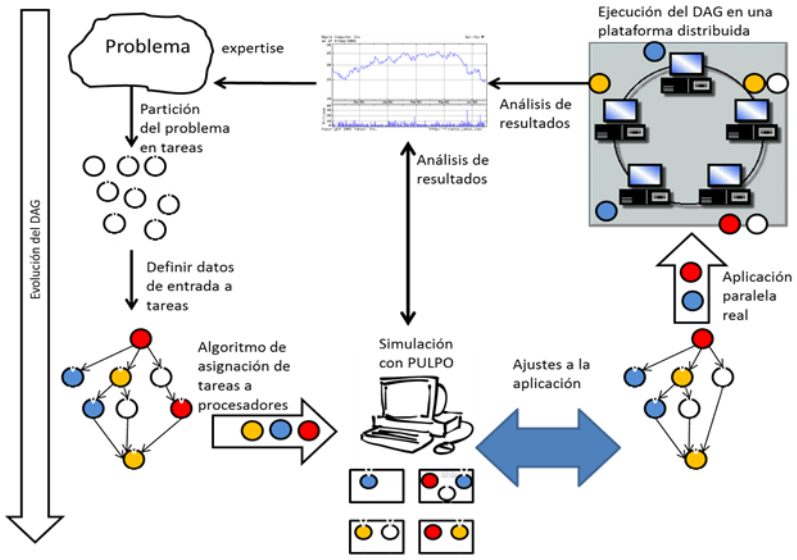


Figura 1. *Ciclo de vida de una aplicación paralela.*

Antes de emprender la ardua labor de implementar la aplicación en real, es necesario recurrir a la simulación para encontrar patrones repetitivos en ambientes controlados, que permitan entender, evaluar y optimizar el desempeño la aplicación. En este sentido, se hace necesario desarrollar pulpo, una herramienta de simulación para entender y evaluar el desempeño de algoritmos heurísticos de asignación de tareas en plataformas distribuidas. El análisis de los resultados de la simulación pudiera ocasionar ajustes en la forma del DAG, de la topología de red ó del algoritmo de asignación de tareas, buscando optimizar el desempeño de la aplicación. Posteriormente, se procede a la implementación real de la aplicación paralela. Por último es importante retroalimentar la experiencia obtenida en el desarrollo de nuevas aplicaciones y mejoramiento de las herramientas de software. Este proyecto busca materializar esta visión con desarrollos tecnológicos propios que nos permitan avanzar en el área. La idea es crear una plataforma web, desde donde un estudiante/investigador de cualquier universidad del país, pueda acceder a artículos relacionados, métodos de enseñanza, descargar de manera gratuita pulpo y recibir asesoría a distancia para implementar sus proyectos.

Definición formal del problema

Esta sección presenta una definición formal del problema de asignación de tareas de un DAG en redes con computadoras heterogéneas y dinámicas, características de las redes actuales que permitirán modelar escenarios realistas. Las definiciones en esta sección están alineadas con la modelación soportada en pulpo.

1 Definición de la Plataforma de Red

Los recursos que componen la red se representan por medio de un grafo no dirigido $PD:: (P, L, \delta, \psi)$ donde P es el conjunto de computadoras disponibles que forman la plataforma, $pi(1 \leq i \leq |P|)$. L es el conjunto de enlaces de comunicaciones que conectan un par de distintos procesadores, $li(1 \leq i \leq |L|)$ *tal que* $l(p_m, p_n) \in L$ denota un enlace de comunicación entre p_m y p_n. Para modelar la naturaleza cambiante en el desempeño de los recursos computacionales se utiliza $\delta::P \rightarrow [0..1]$ que denota el porcentaje de disponibilidad de cada computadora y $\psi:: L \rightarrow Float$ que denota el ancho de banda de cada enlace de comunicación. Esta definición permite considerar el caso extremo en que la disponibilidad de un recurso es igual a cero.

2 Definición de la Aplicación Particionada

La aplicación particionada se representa por medio de un DAG $AP::(V, E, \theta, \tau)$. 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 se define $Pred(v_i)$ 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 vi. Aquella tarea vi tal que $|Pred(v_i)| = 0$ es llamada tarea de entrada y $|Succ(v_i)| = 0$ es llamada tarea de salida. Usamos $\theta::VxV \rightarrow int$ para describir el costo de la transferencia de datos, tal que $\theta(v_i, v_j)$ denota la cantidad de datos a ser transferidos de v_i a v_j. Considerando que los procesadores son heterogéneos, los tiempos estimados de ejecución se representan como $\tau::VxP \rightarrow Int$, donde $\tau(v_i, p_m)$ denota el costo de ejecución estimado de la tarea v_i en el procesador p_m.

3 El Algoritmo de Asignación

Los algoritmos de asignación se enfocan en la generación de una planificación (Scheduling) de tareas en las computadoras de la red, buscando optimizar una función objetivo, que por lo regular es minimizar el tiempo estimado de ejecución (makespan) de la aplicación. La planificación de tareas se puede representar como una función $ASIGNA::V \rightarrow P$, la cual asigna tareas a procesadores. De esta forma, $ASIGNA(v_m, p_j)$ denota que la tarea v_m se asigna al procesador p_j.

4 Mecánica de la Simulación

Pulpo considera que el DAG tiene una *tarea de entrada* y una *tarea de salida*. En caso de que un DAG particular pudiera tener más de una tarea de entrada, este se puede modelar agregando una tarea *dummy* conectada a las diversas tareas de entrada, cuyo costo de cómputo y comunicación sea cero. El mismo proceso se sigue en el caso de que existan varias tareas de salida.

Para coordinar la ejecución de las tareas, utilizamos diferentes tipos de status que permiten determinar la situación de una tarea en cualquier momento a lo largo de la simulación. Los status que puede tener una tarea son los siguientes:

0-creada,
1-asignada,
2-lista para ejecutarse,
3-ejecutándose,
4-pausada,
5-terminada,
6-falla.

La tarea de entrada es la primera en ejecutarse y de ahí comienzan a ejecutarse el resto de las tareas. La tarea de salida es la última tarea en ejecutarse.

Cuando una tarea v_i comienza a ejecutarse en un procesador p_n, se puede utilizar $TI(v_i, P_n)$ y $TF(v_i, P_n)$ para denotar el tiempo de inicio y tiempo final de ejecución de v_i en p_n respectivamente. El *TI* de una tarea de entrada es cero y para las demás tareas se calcula de la siguiente forma.

$$TI\ (v_i, p_n) = max\ \{\ DISP\ (p_n)\ ,\ max_{vk \in Pred(vi)}\ (\ TF\ (\ v_k, p_k) + C(v_k, p_k, v_i, p_n)\ )\ \} \quad (1)$$

Donde en la primer parte de la ecuación se determina $DISP(p_n)$ para indicar 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 recibe la totalidad de los datos transferidos de parte de sus predecesores y por lo tanto se encuentra lista para ejecutarse. Esto se obtiene considerando las tareas predecesoras inmediatas a v_i, el tiempo final (TF) en que estas terminan de ejecutarse en su respectivo procesador p_k y el tiempo que tarda para transferir los datos necesarios de p_k al procesador en consideración p_n. El *TF* de v_i en p_m se determina por

$$TF\ (v_i, p_m) = TI\ (v_i, p_m) + \tau\ (v_i, p_m) \quad (2)$$

que indica que el tiempo final de ejecución de una tarea v_i en p_m equivale al tiempo inicial de ejecución más el tiempo que esta tarda en ejecutarse en el procesador asignado p_m. El *makespan* de la aplicación está determinado por el tiempo de ejecución de v_{salida} (última tarea del DAG).

$$makespan = TF\ (v_{salida}) \quad (3)$$

Resultados

En esta sección utilizamos Pulpo para implementar algunos métodos de asignación de tareas de la literatura y presentamos los resultados obtenidos.

1 Algoritmos de asignación

Para nuestro propósito tomamos de la literatura los algoritmos heurísticos HEFT (Heterogeneous Earliest-Finish-Time) y CPOP (Critical-Path-on-a-Processor). El detalle de ambos algoritmos puede consultarse en (Topcuoglu, 2002). También usamos un método Aleatorio que asigna tareas a procesadores de manera aleatoria, cuidando la precedencia de tareas. Cabe mencionar que para mantener baja la complejidad de un algoritmo, la mayoría de las heurísticas no consideran el tráfico de la red, sino que asumen un número infinito de enlaces de comunicación entre los procesadores. Esto significa que una tarea que finaliza su ejecución, siempre será de capaz de encontrar un enlace de comunicación disponible para transferir datos a sus sucesores. Como se muestra más adelante, esta consideración puede provocar inconsistencias al momento de evaluar los algoritmos.

2 Características de los DAGs

El tamaño de los grafos utilizados (en número de tareas) es de 50 y 100 tareas. En los grafos de 50 tareas el número de predecesores por tarea está en el rango de 0 a 30, mientras que para los grafos de 100 tareas varia en el rango de 0 a 50. La Fig. 2 muestra la relación entre el tamaño del cómputo de las tareas y el tamaño de las comunicaciones entre las tareas de los DAGs. Se observa que existe una cantidad considerable de comunicación entre las tareas de los DAGs. En los DAGs de 50 tareas existe una relación de 5 a 1 entre el tamaño de las comunicaciones y el tamaño del cómputo, mientras que en los DAGs de 100 tareas esta relación es de 7 a 1.

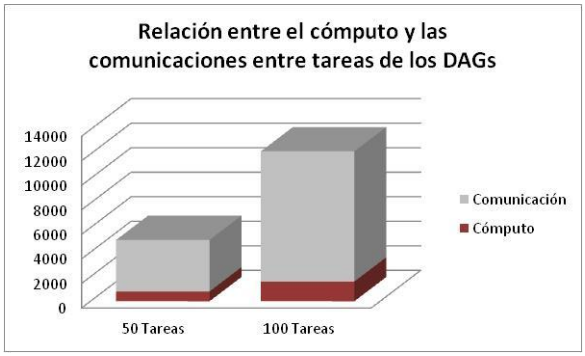


***Figura 2**. Características de los DAGs utilizados en los experimentos.*

3 Desempeño de los algoritmos de asignación

Para evaluar el desempeño de los algoritmos seguimos la siguiente metodología. Los grafos descritos en la Sección 2 fueron utilizados para evaluar los algoritmos HEFT y CPOP tal y como están definidos en (Topcuoglu, 2002). Esto implica la consideración de un número infinito de enlaces de comunicación entre los diferentes procesadores que conforman la plataforma computacional. En nuestros experimentos consideramos escenarios con plataformas computacionales compuestas por 3, 5 y 10 procesadores. Para cada escenario utilizamos 50 grafos de 50 y 100 tareas, dando un total de 300 experimentos. La Fig. 3 muestra los resultados de los experimentos. Se puede observar que en escenarios con 50 Tareas, HEFT es mejor que CPOP para las diferentes plataformas computacionales. Para 3 procesadores, HEFT tiene un 5% de mejor desempeño que CPOP, para 5 procesadores un 18% y para 10 procesadores un 13%. El mismo patrón se puede observar para escenarios con 100 tareas. Para 3 procesadores, HEFT tiene un 2% de mejor desempeño que CPOP, para 5 procesadores un 7% y para 10 procesadores un 12%.

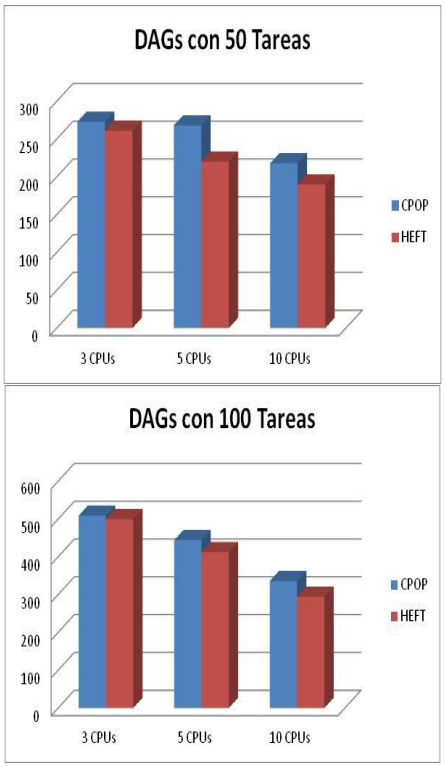


Figura 3. *Desempeño de los algoritmos HEFT y CPOP en ambientes poco realistas.*

Un segundo conjunto de experimentos fueron realizados para evaluar los algoritmos HEFT, CPOP y el método

Aleatorio en Pulpo. Pulpo permite considerar ambientes más realistas, en particular un número de enlaces de comunicación con capacidad finita. En nuestros experimentos consideramos escenarios con plataformas computacionales compuestas por 3, 5 y 10 procesadores. Los procesadores están completamente conectados y cada enlace está configurado para transmitir una unidad de datos por unidad de tiempo. Para cada escenario utilizamos 50 grafos de 50 y 100 tareas, dando un total de 450 experimentos. La Fig. 4 muestra los resultados de los experimentos. Se puede observar que a diferencia de los experimentos anteriores, CPOP tiene un mejor desempeño que HEFT y el método Aleatorio. El método Aleatorio es el que tiene el peor desempeño para todos los escenarios, esto indica que en el contexto de la asignación de tareas a procesadores, es mejor tener un plan de asignación que no tenerlo. Esto debe motivar a los estudiosos del tema a proponer nuevos mecanismos para asignar tareas a procesadores. En escenarios con 50 Tareas y 3 procesadores, CPOP es mejor que HEFT en un 1% y 4% mejor que el método Aleatorio. Cuando se usan 5 procesadores, HEFT tiene un 10% mejor desempeño que CPOP y 5% para el método Aleatorio. Para 10 procesadores HEFT un 27% mejor desempeño que CPOP y 16% que el método Aleatorio.

Para escenarios con 100 tareas tenemos los siguientes resultados. Para 3 procesadores, CPOP tiene un 6% mejor desempeño que HEFT y 9% mejor que el Aleatorio. Para 5 procesadores CPOP es mejor que HEFT y el Aleatorio en un 7% y 23% respectivamente. Mientras que para 10 procesadores, HEFT es ligeramente mejor que CPOP en un 1% y 18% mejor que el Aleatorio.

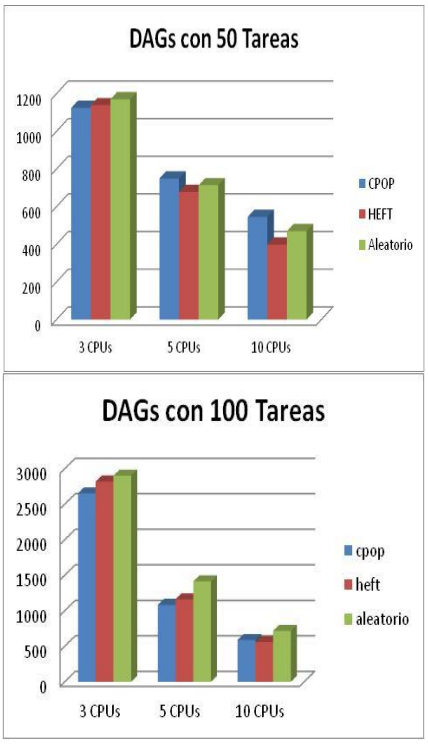


Figura 4. *Desempeño de los algoritmos HEFT y CPOP en Pulpo.*

API de Pulpo

Pulpo fue implementado en phyton utilizando programación orientada a objetos. Tal decisión se debe a que la estructura de phyton simplifica la programación de aplicaciones, permitiendo que estudiantes / investigadores puedan proponer e implementar sus proyectos de una manera rápida.

Un escenario de simulación es definido al crear una instancia de la clase *scenario* con el respectivo nombre del

escenario. La modelación de las tareas del DAG se hace creando instancias de la clase *task* especificando el nombre de la tarea respectiva. Posteriormente el método *addDependancy* permite crear una relación de precedencia entre dos tareas, así como para indicar la cantidad de datos a transmitir entre ambas tareas. El conjunto de las precedencias creadas resulta en el DAG. La modelación de una topología de red particular es a partir de la creación de instancias de la clase *processor* con su respectiva disponibilidad y latencia. Posteriormente se utiliza el método *addLink* para crear un enlace de comunicación entre dos procesadores con su respectivo ancho de banda. La naturaleza cambiante de los recursos en el tiempo se puede modelar de la siguiente manera: En el caso de las computadoras se utiliza el método *addChangeCapacity* cuyo primer parámetro es el tiempo y el segundo parámetro es la capacidad que cambia en el tiempo definido. En el caso de los enlaces de comunicación se tiene el método *addChangeLinkBandwidth* para modelar un cambio en algún punto del tiempo del ancho de banda. Para modelar la asignación de tareas a computadoras heterogéneas se usa el método *addSchedule* de una instancia *task*, donde el primer parámetro es una instancia tipo procesador al cual se asigna la tarea y el segundo parámetro representa el tiempo que tarda la tarea en ejecutarse en el procesador. La simulación se realiza al llamar el método *simulation* de una instancia tipo *scenario*.

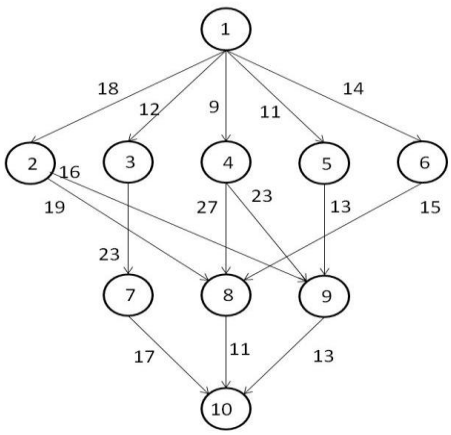
La Fig. 5 muestra un ejemplo con los diferentes elementos que conforman un problema de asignación de tareas a procesadores:

a) El DAG conteniendo las tareas, restricciones de precedencia y transferencias de datos entre tareas.

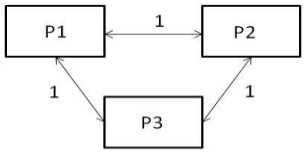
b) La plataforma computacional conformada por procesadores y enlaces de comunicación con su respectiva capacidad y

c) conocimiento de la aplicación reflejada en los tiempos de ejecución estimados de una tarea en los diferentes procesadores.

En este ejemplo particular podemos deducir la heterogeneidad de los procesadores.



(a) DAG



(b) Plataforma computacional

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

Figura 5. *Elementos de un problema de asignación de taras a procesadores.*

La Fig. 6 muestra el uso de pulpo para modelar el ejemplo de la Fig. 5. Se observa la creación de las tareas del DAG con sus respectivas dependencias y transferencias de datos. La topología está compuesta por tres computadoras con sus respectivas capacidades, las cuales están totalmente conectadas por un enlace de comunicación con su respectivo ancho de banda. Para modelar características dinámicas de plataformas computacionales actuales, se incluyen algunos cambios en el desempeño de los recursos a través del tiempo, los cuales se reflejan a lo largo de la simulación. También se muestra la forma de asignar una tarea a una computadora para su posterior ejecución. En este caso consideramos una planificación rígida, sin embargo la planificación de tareas debe ser el resultado de un algoritmo de asignación de tareas.

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

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

''' 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")

''' DAG dependencies '''
dt0 = dag.addDependancy(t1,t2, 18)
dt1 = dag.addDependancy(t1,t3, 12)
dt2 = dag.addDependancy(t1,t4, 9)
dt3 = dag.addDependancy(t1,t5, 11)
dt4 = dag.addDependancy(t1,t6, 14)
dt5 = dag.addDependancy(t2,t8, 19)
dt6 = dag.addDependancy(t2,t9, 16)
dt7 = dag.addDependancy(t3,t7, 23)
dt8 = dag.addDependancy(t4, t8, 27)
dt9 = dag.addDependancy(t4,t9, 23)
dt10 = dag.addDependancy(t5,t9, 13)
dt11 = dag.addDependancy(t6,t8, 15)
dt12 = dag.addDependancy(t7,t10, 17)
dt13 = dag.addDependancy(t8,t10, 11)
dt14 = dag.addDependancy(t9,t10, 13)

''' processors '''
p1=net.addProcessor("p1", 1.0, 0.0)
p2=net.addProcessor("p2", 1.0, 0.0)
p3=net.addProcessor("p3", 1.0, 0.0)

''' network '''
L1 = net.addLink(p1,p2, 1.0)
L2 = net.addLink(p1,p3, 1.0)
L3 = net.addLink(p2,p3, 1.0)

''' changes in capacity over time '''
p1.addChangeCapacity(100, 2)
L1.addChangeLinkBandwidth(150,3)
L3.addChangeLinkBandwidth(200,3)

''' schedule '''
t1.addSchedule(p2,16)
t2.addSchedule(p2,19)
t3.addSchedule(p1,11)
t7.addSchedule(p1,7)
t4.addSchedule(p3,17)
t5.addSchedule(p2,13)
t9.addSchedule(p2,12)
t6.addSchedule(p3,9)
t8.addSchedule(p3,14)
t10.addSchedule(p2,7)

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

***Figura 6**. Ilustración de la API de Pulpo*

Conclusiones y trabajo futuro

En este artículo se expuso la necesidad de contar con un ambiente colaborativo para el estudio y comprensión del cómputo paralelo entre las universidades de Latinoamérica. Se describió a pulpo, una herramienta de simulación para evaluar algoritmos de asignación de tareas en plataformas distribuidas. Pulpo provee funcionalidades y abstracciones necesarias para la implementación y evaluación de algoritmos de asignación en plataformas computacionales con características actuales. Resaltamos la importancia de pulpo al mencionar como algoritmos de la literatura utilizan escenarios poco realistas y esto puede ocasionar inconsistencias en las evaluaciones del desempeño de la aplicación. Esfuerzos futuros se van a centrar en cuatro aspectos: La confiabilidad de los resultados, depuración de posibles errores, colaborar con la comunidad académica interesada en el área y buscar patrones que nos ayuden a entender las posibles divergencias de una aplicación simulada en pulpo y ejecutada en ambientes reales. Así mismo una extensión del proyecto busca desarrollar un conjunto de librerías que permitan desarrollar aplicaciones paralelas en redes trabajando con Windows.

Referencias

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

Agarwal, T., Sharma, A., and Kale, L (2006). Topology-aware task mapping for reducing communication contention on large parallel machines. IEEE/IPDPS, page 10 pp. ..

Beaumont, O., Legrand, A., Marchal, L., and Robert, Y. (2005). 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.

Chervenak, A., Foster, I., Kesselman, C., Salisbury, C., & Tuecke, S. (2000). The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. Journal of network and computer applications, 23(3), 187-200.

Eshaghian, M. and Wu, Y. (1997). Mapping heterogeneous task graphs onto heterogeneous system graphs. In Proceedings of Heterogeneous Computing Workshop (HCW'97), pages 147–160,

Eun-Kyu B., Yang-Suk K., Jin-Soo K., Deelman, E. (2011). BTS: Resource capacity estimate for time-targeted science workflows, Journal of Parallel Distrib. Comput. (JPDC), 71(6): 848-862

Foster, I., Kesselman, C., & Tuecke, S. (2001). The anatomy of the grid: Enabling scalable virtual organizations. International journal of high performance computing applications, 15(3), 200-222.

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

GridSim. (2010). The GridSim project homepage, En línea: http://www.cloudbus.org/gridsim/.

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

Hernandez, I. and Cole, M. (2007b), 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.

Kwok, Y. and Ahmad, I. (1999). Static algorithms for allocating directed task graphs to

multiprocessors, ACM Computing Surveys, 31(4):406–471.

Mell, P., & Grance, T. (2011). The NIST definition of cloud computing. National Institute of Standards and Technology Special Publication 800-145, 7 pages.

Olteanu A., Pop F., Dobre C. (2011) Re-scheduling and Error Recovering algorithm for Distributed Environments .Sci. Bull., Series C., 73(1).

Olteanu A., Pop F., Dobre C (2012). A dynamic rescheduling algorithm for resource management in large scale dependable distributed systems. Elsevier Computers & Mathematics with Applications, 63(9): 1409-1423.

Pacheco P. (2011), An Introduction to Parallel Computing. ISBN-10: 0123742609, Morgan Kaufmann, 1st edition.

Razdan S. (2014, August). Fundamentals of Parallel Computing. ISBN-10: 1842658808, Alpha Science International Ltd, 1st edition.

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

Sven K., Riddle S. and Zinn D. (2011). Improving Workflow Fault Tolerance through Provenance-Based Recovery. Scientific and Statistical Database Management, Lecture Notes in Computer Science 6809: 207-224

Topcuoglu, H. (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Transactions on Parallel and Distributed Systems, 13(3):260–274.

Vouk, M. A., & Mouallem, P. A. (2011, November). On high-assurance scientific workflows. In High-Assurance Systems Engineering (HASE), 2011 IEEE 13th International Symposium on (pp. 73-82). IEEE.